Analyse et détection de logiciels de rançon

Aurélien Palisse

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
COMUE **U**NIVERSITE **B**RETAGNE **L**OIRE

Ecole Doctorale N°601
Mathèmatique et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

« Aurélien PALISSE »

« Analyse et détection de logiciels de rançon »

Thèse présentée et soutenue à RENNES, le 4 mars 2019

Unité de recherche : Institut National de Recherche en Informatique et en Automatique (Inria)

Rapporteurs avant soutenance :

Christophe Clavier, Professeur, Université de Limoges

Peter Ryan, Professeur, Université du Luxembourg

Composition du jury :

Président :	Davide Balzarotti, Professeur, Eurecom
Examinateurs :	Christophe Clavier, Professeur, Université de Limoges
	Peter Ryan, Professeur, Université du Luxembourg
	Colas Le Guernic, Ingénieur, Sekoia
	Hélène Le Bouder, Maître de conférence, IMT Atlantique
Dir. de thèse :	Jean-Louis Lanet, Chercheur, Inria

Table des matières

Introduction

Depuis la fin des années 70 et le début des années 80, et l'arrivée des premiers ordinateurs personnels, les us et coutumes dans le monde professionnel, mais aussi privé se sont transformés. Nous comptons ainsi près d'un foyer sur deux équipé d'un ordinateur en 2017 dans le monde [61]. Dans le même temps, l'Internet s'est développé et compte aujourd'hui plus de 4 milliards d'utilisateurs, soit environ 40% de la population mondiale [57]. De nouvelles menaces sont donc apparues de pair avec les nouveaux usages, nous entendons : les logiciels malveillants. La définition est donnée : "*un logiciel malveillant est un programme développé dans le but de nuire à un système informatique, sans le consentement de l'utilisateur dont l'ordinateur est infecté*" [23]. Le sujet de cette thèse concerne un type particulier de logiciels malveillants appelés logiciels de rançon. Ceux-ci sont : "*des logiciels malveillants qui prennent en otage des données personnelles*" [22]. L'objectif, bien entendu, est d'obtenir en retour de la restitution des données un paiement.

La première attaque documentée d'un logiciel de rançon date de 1989. Dans les décennies qui ont suivi, peu de cas similaires ont été rapportés. Il a fallut attendre l'année 2013 avec *CryptoLocker* pour que les logiciels de rançon reviennent sur le devant de la scène. Nous avons ensuite constaté une hausse quasi-continue du nombre d'attaques et de victimes jusqu'à aujourd'hui en 2018. De nombreux symboles de nos sociétés ont été victimes de ces attaques : des hôpitaux [56], des universités, des postes de police, des ministères [11], une centrale nucléaire [51] et de nombreuses entreprises. Les campagnes les plus célébres sont celles de *WannaCry* et de *Petya* en mai et juin 2017 respectivement [55]. Les autorités gouvernementales en France ont donc mis en place des moyens de sensibilisation pour endiguer le phénoméne, notamment l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI). En effet, les enjeux financiers sont considérables. L'ANSSI conseille par ailleurs de ne pas payer la rançon [53].

Cette thèse a débuté en fin d'année 2015 et coïncide avec une hausse de 300% des attaques entre l'année 2015 et 2016 [49]. Les logiciels de rançon deviennent même la charge utile numéro un des logiciels malveillants. Au moment où j'écris ces mots, les attaques sont en baisse [58]. Néanmoins, les logiciels de rançon constituent toujours une menace de premier plan.

L'objectif de la thèse est de proposer une ou des contre-mesures aux logiciels de rançon. Celle-ci est conçue pour être temps réel et déployée sur des ordinateurs personnels, comme un antivirus classique. Au début de la thèse, très peu de littérature est consacrée aux logiciels de rançon. Le succès des logiciels de rançon s'explique aussi en partie par le fait que les compagnies d'antivirus ont mis du temps à prendre la mesure de la menace. Nous présentons principalement dans la revue de littérature des contre-mesures. Les contributions qui suivent, plus hétéroclites, peuvent

néanmoins intéresser le lecteur : (1) Hernandez-Castro *et al.* [82] font une analyse économique du modèle d'affaire lié à la rançon, (2) Huang *et al.* [83] s'intéressent au circuit de paiement et (3) Bar-On *et al.* [64] discutent des stratégies de sauvegarde.

Plan de la thèse

Le chapitre 1 introduit les notions indispensables pour la compréhension du lecteur. Un état de l'art des contributions du domaine est présenté dans le chapitre 2. Le chapitre 3 met en avant notre propre plateforme d'analyse automatique de logiciels malveillants : Malware - O - Matic. Les chapitres 4, 5 et 6 présentent les contributions de cette thèse. À la suite d'une première série d'expérimentations, une contre-mesure est proposée chapitre 4. Les chapitres 5 et 6 proposent une seconde contre-mesure plus aboutie et bien plus efficace. Les travaux en cours et futurs sont ensuite présentés chapitre 7. Finalement, nous terminons avec la conclusion générale de la thèse.

Chapitre 1

Prérequis

Ce chapitre introduit des notions essentielles concernant le système d'exploitation Windows. Nous introduisons les termes suivants : ring 0 et 3, "minifilter driver", crochet ou "hook", injection de code, pilote de filtre, bac à sable, etc. Ceux-ci sont nécessaires au lecteur pour comprendre les idées présentées dans les chapitres suivants. Nous présentons également des méthodes d'analyse dynamique de programmes malveillants et justifions leurs emplois au détriment des méthodes statiques. Le chapitre présente succinctement des notions pour la plupart complexes et pour lesquelles les livres suivants donnent des explications bien plus complètes : [67], [107] et [100].

1.1 Architecture générale

Dans cette thèse nous ne nous intéressons pas aux aspects matériels, mais uniquement à la couche "logicielle". Le modèle d'architecture de Von Neumann s'applique aux ordinateurs considérés ici. Nos contributions considèrent le système d'exploitation Windows. Mais il est tout à fait possible de transposer nos contributions vers une autre plateforme logicielle. Windows à travers ses différentes versions est le système le plus utilisé [1], celui-ci est donc une cible de choix pour les attaquants. Le système d'exploitation est en fait une couche d'abstraction sur le matériel. Celui-ci est capable de gérer différents types de matériels de manière transparente. De manière similaire le système offre une couche d'abstraction pour les applications utilisateurs (e.g., accès disque).

1.1.1 Ring 3 et ring 0

Le système d'exploitation Windows offre deux niveaux de privilèges : le mode utilisateur et le mode noyau. Le positionnement de bits dans des registres matériels et des structures de données du système permettent de différencier ces deux modes et de gérer les accès aux ressources matérielles (e.g., mémoire) ou encore l'exécution d'instructions privilégiées (e.g. HLT). Les deux modes présentés ci-dessus n'ont rien à voir avec la gestion des droits et permissions des utilisateurs sur Windows. Le code qui s'exécute sur le processeur peut fonctionner selon ces deux modes.

1. https://netmarketshare.com/operating-system-market-share.aspx

Il en est de même pour la gestion de l'espace mémoire. La mémoire physique est accessible par des adresses virtuelles qui sont résolues par le système et le matériel. Les adresses virtuelles sont découpées en deux grand ensembles : espace utilisateur et espace noyau. Normalement le code qui s'exécute en mode noyau est présent dans l'espace noyau et réciproquement pour le mode utilisateur. La FIGURE 1.1 présente les quatre niveaux de privilèges qui peuvent être affectés à une opération. Seuls deux anneaux de protection sont utilisés par Windows : le ring 3 et le ring 0. Une des raisons à cela est la compatibilité entre les différentes architectures matérielles. Le processeur empêche une application qui s'exécute dans un anneau "extérieur" d'accéder à des ressources dans un anneau "intérieur". Une violation des niveaux de privilèges déclenche une exception de la part du processeur qui peut être de deux types : faute de page ou faute de protection générale.

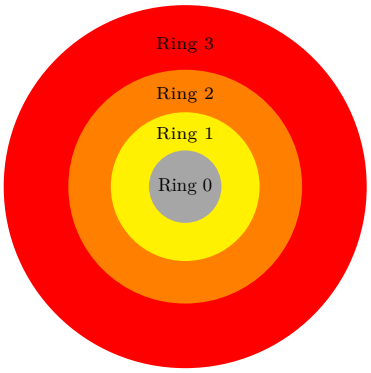

FIGURE 1.1 – Les anneaux de protection définis par le système d'exploitation Windows. Le ring 3 correspond au mode utilisateur, le ring 2 au mode utilisateur protégé, le ring 1 au mode noyau peu sensible et ring 0 au mode noyau.

1.1.2 Windows API, native API et noyau

Les applications fonctionnent pour la plupart dans l'espace utilisateur, à l'exception des pilotes et du noyau. Les logiciels de rançon au meilleur de ma connaissance s'exécutent exclusivement dans l'espace utilisateur [2]. Chaque application dispose de son propre espace mémoire, de ses ressources et permissions. Si une application utilisateur exécute une instruction non valide et devient défaillante, alors le système reprend la main et peut terminer la tâche. Les applications ne peuvent accéder directement à du matériel et certaines instructions leur sont interdites. L'accès aux ressources du système doit passer par ce que l'on appelle la Windows API ou la Native API. La Windows API est un ensemble de librairies qui offre aux développeurs des interfaces pour accéder au système de fichiers, réseaux, registre, etc. La native API est une interface un peu plus bas niveau qui est principalement utilisée par les logiciels malveillants. Celle-ci ne doit normalement pas être employée, car non documentée, et offre des fonctionnalités supplémentaires. La pile des appels pour accéder à un fichier par exemple, conduit à un transfert du flot d'exécution du programme vers une fonction du noyau : un appel système. Ce transfert est réalisé via des

2. une exception : les logiciels de rançon qui ciblent le secteur de démarrage (e.g., Petya)

instructions spécifiques (e.g., SYSENTER) qui permettent à un fil d'exécution d'acquérir les droits nécessaires. Un fil d'exécution utilisateur fait des transitions ou des "aller-retours" entre l'espace utilisateur et l'espace noyau en fonction de ses appels systèmes. Le code qui s'exécute dans l'espace utilisateur n'y est donc pas confiné. La FIGURE 1.2 présente le flot d'exécution d'un programme utilisateur qui accède en lecture à un fichier. Celui-ci peut utiliser la Windows API et la librairie *kernel32.dll* qui contient la routine nécessaire ou la native API qui possède une routine équivalente. Le flot d'exécution du programme est ensuite transféré au noyau.

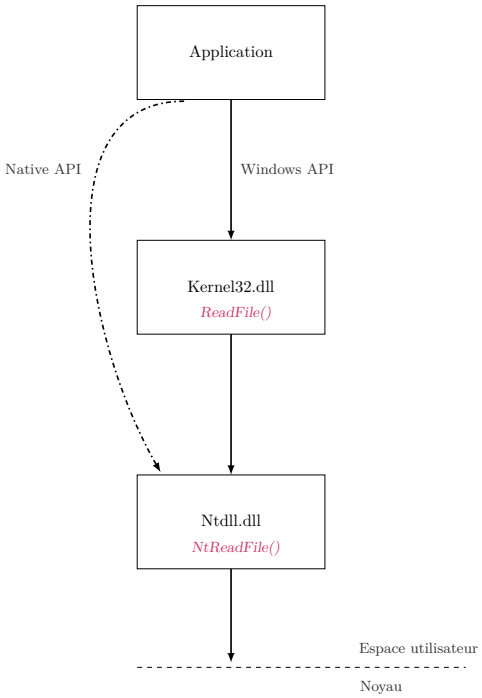

FIGURE 1.2 – Un exemple d'une application utilisateur qui accède en lecture à un fichier. Deux méthodes proposées : Windows API ou Native API.

Contrairement aux processus qui s'exécutent dans l'espace utilisateur, les processus du noyau partagent le même espace mémoire. La gestion des exceptions est différente, l'exécution d'une instruction invalide entraîne un plantage de la machine (i.e., blue screen). Le code qui s'exécute en mode noyau peut réaliser à peu près toutes les opérations qu'il est possible de réaliser sur la machine, et même manipuler du code utilisateur. Le noyau Windows (e.g., *NtosKrnl.exe*) fournit des interfaces pour les pilotes qui s'exécutent dans l'espace noyau, pour allouer de la mémoire par exemple. Un pilote est en fait un programme informatique dans l'espace noyau qui s'exécute avec le plus haut niveau de privilège, c'est-à-dire en "ring 0". Un pilote peut interagir avec du matériel ou simplement ajouter une couche logicielle (e.g., chiffrement disque). Des pilotes fournis par Windows ou des tiers, de type "export driver" peuvent également offrir des interfaces plus spécifiques (e.g., primitives cryptographiques, *cng.sys*). La plupart des produits de sécurité se positionnent dans le noyau. Cette position bas niveau leur permet de surveiller les applications utilisateurs tout en bénéficiant de davantage de privilèges.

1.1.3 Crochet ou "hook"

Le format *Portable Executable* [59] est utilisé par les exécutables Windows, les librairies, pilotes, etc. Ce format contient dans son en-tête des structures de données qui permettent au PE Loader de charger en mémoire puis d'exécuter le binaire. Un fichier PE contient ensuite différentes parties ou sections dont une qui contient du code, mais aussi des données, des ressources et des structures particulières telle que *l'Import Address Table* (IAT). Cette structure fait la correspondance entre les fonctions importées par l'exécutable dans des librairies dynamiques et leurs adresses. Le PE Loader est responsable au chargement de l'importation des librairies dans l'espace mémoire du processus puis de la résolution des adresses via l'IAT. Les adresses des fonctions ne sont donc pas statiques dans ce cas. Seule une édition de lien avec une librairie statique fournit des adresses statiques, le code de la librairie est alors ajouté au code de l'application. La différence entre les deux est alors non triviale, car rien n'indique dans l'en-tête d'un PE que celui-ci utilise une librairie statique. Il est également possible d'importer des fonctions non listées dans l'en-tête d'un PE en réalisant une édition de lien en temps réel, c'est-à-dire "runtime linking". Pour cela, les fonctions utilisées sont principalement : *LoadLibrary* et *GetProcAddress*. Ces deux fonctions permettent d'importer n'importe quelle fonction présente dans une librairie dynamique sur le système quand on le souhaite. De plus, le fait de savoir qu'elles sont utilisées ne permet pas de connaître systématiquement les fonctions qui sont importées (e.g., offuscation). Les logiciels malveillants utilisent les librairies dynamiques principalement dans trois cas :

1. API Windows pour accéder à des ressources du système, par exemple *kernel32.dll*,
2. API tierce pour interagir avec un programme particulier,
3. stocker du code malveillant, par exemple s'attacher à un processus légitime.

Il est possible de modifier le flot de contrôle d'un programme dans l'espace utilisateur à l'aide d'un crochet, plus communément appelé "hook". Un programme malveillant peut ainsi cacher des fichiers, des processus, des connexions réseaux, etc, à un ou plusieurs de ses pairs. Pour cela, le programme malveillant doit accéder à l'espace mémoire du processus victime et modifier une ou plusieurs adresses en mémoire virtuelle. Une des méthodes consiste à modifier l'IAT afin d'intercepter l'appel à la fonction légitime souhaitée puis de transmettre celle-ci vers une routine que contrôle l'attaquant. Cette technique est facilement détectable et des techniques bien plus évoluées existent. Une autre technique appelée "inline hook" va directement modifier le code de la fonction cible dans la librairie dynamique. L'objectif est de modifier son prologue. Pour mettre en place le crochet, il faut attendre que la librairie soit chargée dans l'espace mémoire du processus victime. Le programme malveillant peut ensuite mettre en place un saut vers sa propre routine, mais aussi modifier le comportement de la fonction légitime. La plupart des programmeurs qui développent des produits de sécurité s'attendent à ce qu'un programme malveillant modifie les premiers octets de leur routine. C'est pourquoi, les attaquants tentent de placer le saut à une position plus avancée dans la fonction victime du crochet afin de ne pas être détecté.

En reprenant le même exemple que la FIGURE 1.2, nous pouvons ajouter un crochet, FIGURE 1.3, qui permet de détourner le flot d'exécution d'un programme vers du code contrôlé par un attaquant. Pour ce faire, l'attaquant peut via l'injection d'une librairie malveillante, *Malicious.dll*, dans l'espace mémoire du processus utilisateur détourner le flot d'exécution. Il contrôle

ainsi les paramètres de la fonction, la valeur de retour, etc. Un tel crochet peut aussi être utilisé par des produits de sécurité pour vérifier si une application détourne une API de son utilisation légitime. Les produits de sécurité conscients que les attaquants emploient la native API, se servent alors d'un pilote ou de crochets sur *ntdll.dll* pour protéger le système.

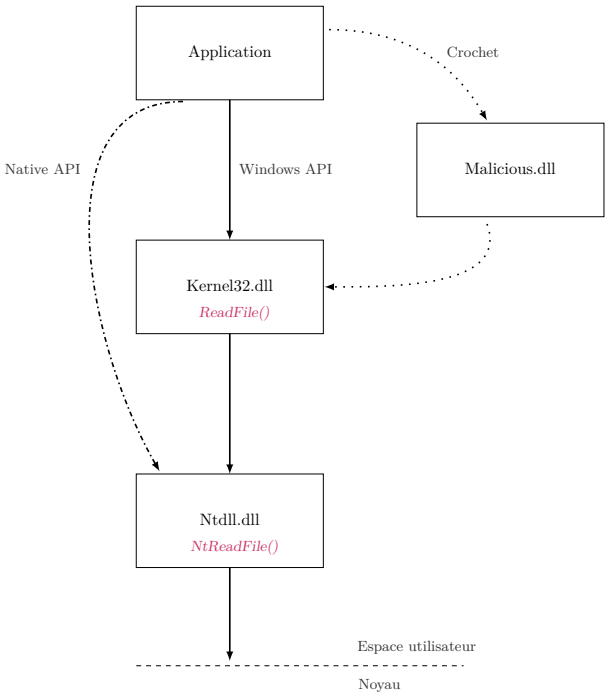

FIGURE 1.3 – Le même exemple que FIGURE 1.2 excepté : présence d'un comportement malveillant. Un crochet peut également intercepter l'appel à la fonction *ReadFile* et détourner le flot d'exécution vers *Malicious.dll*.

La librairie *Detours* [85] développée initialement par Microsoft Research en 1999 offre la possibilité aux développeurs d'intercepter des appels à l'API Windows grâce à des crochets. Depuis 1999 et le support de IA32, ces capacités ont été étendues aux architectures ARM et IA64. Cette librairie est basée sur la méthode "inline hook" et permet d'instrumenter du code, de modifier son comportement, etc. Celle-ci est largement utilisée par des produits de sécurité, des logiciels malveillants, mais aussi dans le domaine de la recherche pour réaliser des preuves de concepts. Son utilisation est très aisée, mais son usage n'est pas furtif car largement répandu.

Les crochets présentés ci-dessus sont "locaux", ils manipulent des messages à destination du processus dans lequel ils se trouvent. La fonction *SetWindowsHookEx* est capable de réaliser cela sur des processus distants. En fait, ce sont des événements particuliers définis par la Windows API [60] qui autorisent la mise en place de tels crochets. L'attaquant ne s'intéresse pas nécessairement à l'événement qu'il surveille. Il peut utiliser cette fonction uniquement pour accèder à l'espace mémoire du processus victime. La fonction cible tous les fils d'exécution du système ou seulement un seul. Nous pouvons également voir cela, comme un moyen d'injecter du code.

1.1.4 Injection de code malveillant

Nous avons vu que l'API Windows est une interface nécessaire pour les applications en ring 3 afin d'intéragir avec le matériel mais aussi le système. La vision que les applications légitimes ont du système peut être abusé par des crochets mis en place par un programme malveillant présent lui aussi en ring 3. Seulement, pour mettre en place les crochets celui-ci doit être capable d'accèder à l'espace mémoire de l'application victime. Pour cela, le programme malveillant doit disposer des droits administrateurs ou être capable d'acquérir les droits nécessaires. En effet, la manipulation de certaines des APIs qui permettent d'injecter du code est restreinte. L'injection de code permet au programme malveillant de s'exécuter dans le contexte de l'application légitime et ainsi d'évader plus facilement la détection. L'injection de code malveillant dans un processus légitime peut être réalisée par différentes techniques, par exemple :

1. injection d'une librairie dynamique ; "dll injection",
2. ajout de code directement dans l'espace mémoire d'un processus ; "direct injection".

La première méthode consiste à forcer le chargement de la librairie malveillante via l'exécution d'un fil d'exécution contrôlé par l'attaquant dans le processus victime. Pour cela, la fonction *CreateRemoteThread* est généralement utilisée. Une fois la librairie malicieuse chargée, le système passe la main au point d'entrée de celle-ci, la fonction *Dllmain*, qui peut ensuite déclencher le comportement malveillant.

La seconde méthode est utilisée pour injecter du code déjà compilé (i.e., comme précédemment) ou du "shellcode", c'est-à-dire principalement de l'assembleur sous forme de chaîne de caractère. Pour cela, il est nécessaire d'allouer de la mémoire dans le processus victime puis d'écrire à l'adresse souhaitée. Les fonctions *VirtualAllocEx* et *WriteProcessMemory* sont alors utilisées. Une fois le code en place, la fonction *CreateRemoteThread* est appelée à l'adresse de base choisie pour déclencher le comportement malveillant. Cette méthode est plus complexe à mettre en place, et peut affecter la stabilité du code légitime. Le contexte dans lequel le code malveillant s'exécute est particulier, la résolution des chaînes de caractères, mais aussi des fonctions importées doit être gérée spécifiquement.

Une fonctionnalité de Windows peut également être abusée par les logiciels malveillants pour s'injecter dans un programme. Il s'agit de la technique "AppInit DLLs". En fait, deux clés de registre spécifient les librairies dynamiques qui sont chargées par *user32.dll* lorsque cette dernière est chargée par un processus. En pratique, un grand nombre de programmes utilisent cette librairie. La modification des clés de registres permet donc à un programme malveillant sous la forme d'une librairie dynamique de s'exécuter dans le contexte des programmes utilisateurs qui utilisent *user32.dll*, et ce de manière persistante. Ce mécanisme peut être désactivé à partir de Windows 8.

1.2 Gestion du système de fichiers

Nous présentons le fonctionnement du système de fichiers, car il s'agit de la cible principale des logiciels de rançon. Nous appelons un système de fichiers [3] : "*une façon de stocker les informations et de les organiser dans des fichiers sur ce que l'on appelle des mémoires secondaires, soit des supports non volatile*". Les chapitres 2, 5 et 6 de cette thèse tirent partie des notions introduites ici pour proposer des contre-mesures.

L'accès au système de fichiers et au support de stockage est assuré par différents composants, la plupart sont présents dans l'espace noyau. Le noyau Windows est composé des trois éléments suivants : (1) Hardware Abstraction Layer (HAL), (2) le noyau et (3) un ensemble de services appelé Windows NT Executive.

Le premier élément, HAL fournit une couche d'abstraction sur le matériel. HAL est portable, gère différentes architectures matérielles et exporte les fonctions nécessaires au noyau.

Le noyau quant à lui offre les fonctionnalités ou "primitives" essentielles au fonctionnement du système d'exploitation. Il exporte des fonctions qui sont ensuite utilisées par le Windows NT Executive, pour fournir des services plus haut niveau. Le noyau est responsable en autres choses de la planification des processus et des fils d'exécution, de la synchronisation des ressources partagées, de la gestion des interruptions, etc.

Le Windows NT Executive offre de nombreux services ou sous-modules en s'appuyant sur HAL et le noyau. Le gestionnaire de la mémoire virtuelle, des I/O, du cache et des objets en font partie à titre d'exemple. Un pilote attaché au système de fichiers est associé à cet ensemble.

Le système d'exploitation Windows à travers le gestionnaire des objets, associe à chaque type abstrait (e.g., *EPROCESS*) des services et des méthodes. Les types sont pour la plupart des structures opaques pas toujours documentées et manipulées par le module noyau correspondant. Les fichiers sont représentés par des objets, mais aussi les fils d'exécution, les pilotes, etc. Un "handle" est en fait une référence sur une instance d'un objet pour un processus donné, par exemple l'ouverture d'un fichier par une application utilisateur.

1.2.1 Espace utilisateur

La Windows API offre la possibilité à une application utilisateur d'accéder au système de fichiers et ainsi de sauvegarder des données sur un support non-volatile. Trois méthodes sont principalement utilisées : (1) lecture/écriture via des appels systèmes successifs, (2) chargement des données en mémoire virtuelle (i.e., file-mapping) et (3) même chose que (1) avec mise en cache. Le flot d'exécution va donc passer de l'espace utilisateur à l'espace noyau dans le premier cas, et cela, à un coût. Par exemple, lors d'une requête en lecture, le pilote du système de fichiers doit copier les données dans l'espace mémoire du noyau puis les copier à nouveau dans le tampon alloué par l'application. Les performances ne sont donc pas optimales. La seconde méthode permet de "mapper" le contenu d'un fichier dans l'espace mémoire d'un processus, c'est-à-dire en mémoire virtuelle. Le contenu du fichier n'est pas nécessairement résidant en mémoire volatile, quand ce n'est pas le cas, une faute de page est générée pour charger les données. L'avantage

3. paraphrase Wikipedia

ici, est que si plusieurs processus ouvrent le même fichier alors ceux-ci partagent la même "vue" (i.e., adresses physiques) et les modifications sont visibles immédiatemment. De plus, le fichier peut être modifié directement en mémoire et les modifications sur le support non volatile seront appliquées par un fil d'exécution asynchrone du système. L'avantage est que les requêtes en écriture peuvent ainsi être regroupées pour de meilleurs performances. Des fonctions distinctes sont utilisées pour (1) et (2) mais présentes dans la librairie *kernel32.dll*, par exemple, *ReadFile* et *MapViewOfFile* respectivement. La troisième possibilitée est identique à (1) pour l'API mais exploite le mécanisme (2) de manière transparente pour l'application. C'est le système via le gestionnaire de cache qui fait appel au gestionnaire de la mémoire virtuelle pour allouer l'espace nécessaire (i.e., file-mapping) et résoudre les fautes de page. Cette méthode met en cache des données en mémoire virtuelle dans l'espace du noyau et non du processus. De plus, les requêtes sur un fichier n'ont pas systèmatiquement besoin d'atteindre le matériel. En effet, au premier défaut de page un mécanisme appelé "read-ahead" va charger en mémoire physique davantage de données que demandé et ainsi anticiper sur les futurs besoins. De manière similaire à (2), une modification du contenu d'un fichier en mémoire volatile est répercuté via un fil d'exécution système asynchrone en mémoire non-volatile.

1.2.2 Espace noyau

La FIGURE 1.4 présente les composants qui gèrent les entrées-sorties sur le matériel. Dans l'espace noyau, le composant essentiel pour nous est le gestionnaire des I/O, toutes les requêtes d'entrées-sorties transitent par lui. Il se situe au sommet de l'architecture en "couche", visible FIGURE 1.4. Sa conception mais aussi celle des sous-modules attachés au système de fichiers (e.g., pilote) est basée sur des paquets appelés I/O request packet (IRPs). Toutes les requêtes d'entrées-sorties sont transmises via ce mécanisme [4]. Celles-ci sont habituellement créées par le gestionnaire des I/O en réponse à une requête provenant d'une application utilisateur. L'IRP ainsi créée, est ensuite transmise au gestionnaire des filtres, qui lui-même la transmet au pilote du système de fichiers correspondant, et ainsi de suite vers le matériel. À tout moment l'IRP peut être annulée ou retournée incomplète par un pilote ou module dans cette architecture en "couche". Les pilotes et sous-modules du Windows NT Executive peuvent initier leur propre IRP. Une IRP est un conteneur qui contient des informations comme : le niveau de privilège du processus à l'origine de la requête, le tampon associé à celle-ci, son statut, ses drapeaux (e.g., mise en cache), etc. Il existe une exception où une entrée-sortie peut utiliser un autre format qu'une IRP, il s'agit des requêtes sur un fichier mis en cache. Dans ce cas, une FAST I/O est générée par le gestionnaire des I/O puis dirigée vers le gestionnaire de cache, sans que celui-ci interroge nécessairement le matériel. La mise en cache et le "mappage" en mémoire virtuelle d'un fichier, présenté précédemment, entraînent inévitablement des fautes de page. Celles-ci sont résolues par des IRPs appelées "non-cached paging I/O", car elles ne peuvent déclencher à nouveau une faute de page, sinon le système tombe en panne. Nous distinguons donc trois types de requêtes :

1. non-cached I/O (e.g., IRP),
2. non-cached paging I/O (e.g., faute de page),

4. une exception : les FAST I/O

3. cached I/O (e.g., FAST I/O).

Le gestionnaire de la mémoire virtuelle, des I/O et du cache travaillent tous les trois conjointement pour assurer la gestion des entrées-sorties et plus généralement du système d'exploitation.

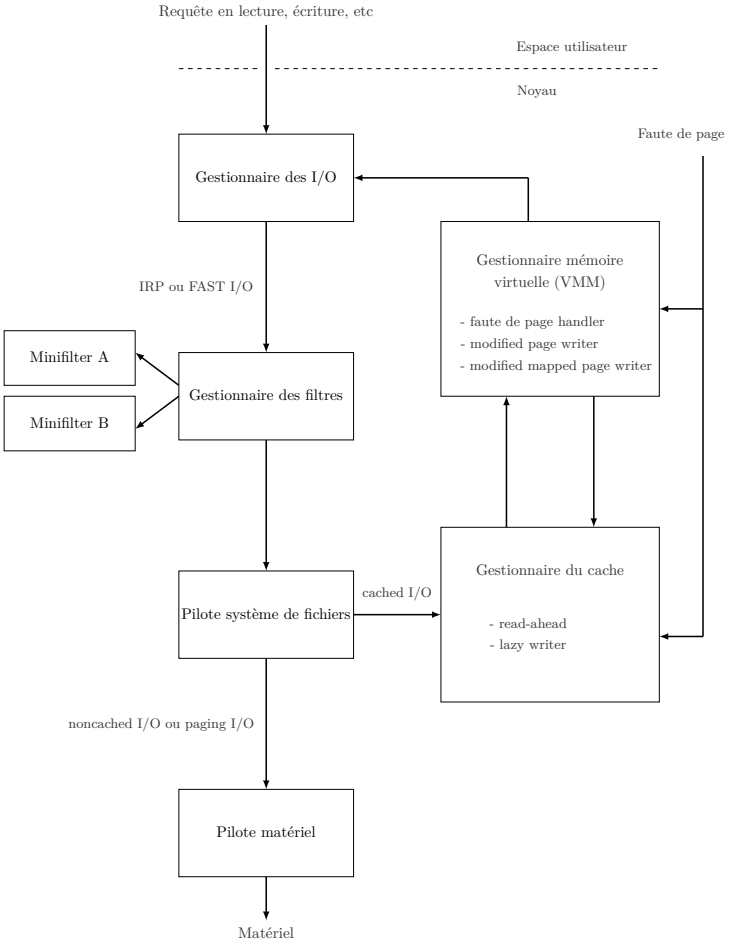

FIGURE 1.4 – Architecture du noyau Windows concernant la gestion des entrées-sorties sur les supports de stockage. Le gestionnaire des filtres gère les pilotes de filtre qui sont attachés à un support. Ceux-ci disposent d'une altitude (e.g., chiffrement) dans la "pile".

Pour surveiller les requêtes d'entrées-sorties à destination des supports non-volatiles, l'architecture en "couche" présentée en FIGURE 1.4, dispose d'un type particulier de pilotes appelés : "file system minifilter driver" ou pilote de filtre. Ces derniers s'insérent dans cette hiérarchie pour proposer des services qui ne sont pas disponibles à partir des pilotes proposés initialement. Ils ont la capacité d'intercepter toutes les requêtes, c'est-à-dire en lecture, écriture, etc, et de les modifier ou bloquer. Ce type de pilote est utilisé par des produits de sécurité, de sauvegarde, de chiffrement, etc. En fonction de la nature des services que les pilotes de filtre procurent, ceux-ci sont positionnés à une "altitude" prédéfinies par le gestionnaire des filtres. Par exemple, un pilote de filtre qui implémente un chiffrement du système de fichiers sera à une "altitude" basse afin de ne pas interférer avec un pilote de filtre qui réalise des sauvegardes et donc positionné plus

haut. Dans cette thèse, nous utilisons un pilote de filtre dans les chapitres 5 et 6 pour détecter en temps réel une menace. Certains challenges sont malgrés tout à relever. Microsoft fournit un kit de développement avec le Windows Driver Kit (WDK) qui offre des échantillons de code et une documentation. La difficulté est que le développement d'un pilote doit être minutieux car la moindre erreur entraîne la défaillance de la machine. De plus, un pilote de filtre ne doit pas corrompre les données à destination du système de fichiers. Par ailleurs, la performance est une chose essentielle. L'ajout d'un pilote de filtre augmente la latence des requêtes d'entrées-sorties, et celle-ci doit être la plus faible possible. Il faut donc utiliser de manière parcimonieuse un tel pilote pour ne pas rendre inutilisable la machine. L'utilisation d'un pilote permet à une contre-mesure aux logiciels de rançon de disposer de davantage de privilèges que ceux-ci, mais aussi d'être transparente. Les pilotes de filtre sont portables, peu importe le système de fichiers sur lequel il s'attache (e.g., NTFS), le système d'exploitation offre la couche d'abstraction nécessaire.

1.3 Analyse dynamique de logiciels malveillants

Deux méthodes d'analyse de binaire sont principalement utilisées dans la littérature : (1) dynamique et (2) statique. Une analyse dynamique d'un binaire exécute celui-ci une ou plusieurs fois au moins de manière partielle afin d'observer son comportement. L'environnement dans lequel le binaire est exécuté est contrôlé afin de garantir son confinement.

À l'inverse une analyse statique a pour objectif d'inférer le comportement du binaire sans l'exécuter. Des techniques permettent de simuler l'exploration du flot de contrôle du programme (e.g., exécution symbolique [91]). Mais celles-ci ne peuvent garantir que tous les sauts soient correctement interprétés ou résolus.

Des avantages et des inconvénients sont donc inhérents aux deux méthodes. Celles-ci sont en fait complémentaires, l'utilisation conjointe des deux techniques est appelée analyse hybride.

Nous préférons dans cette thèse utiliser l'analyse dynamique, pour la simple raison que nous nous intéressons au comportement des logiciels de rançon sur le système de fichiers. Il est bien plus aisé d'observer leurs comportements, c'est-à-dire leurs effets sur le système, et ainsi de mettre au point une contre-mesure dans ce contexte. En effet, nous n'avons pas à gérer les protections éventuelles des binaires. Par protection, nous entendons des techniques plus ou moins évoluées pour cacher le comportement d'un binaire ou simplement compliquer son analyse. De nombreuses techniques peuvent être utilisées (e.g., chiffrement, offuscation) et sont pour la plupart des problèmes d'actualités difficiles à traiter.

Cohen *et al* [70] en 1984 démontrent deux résultats trés importants : (1) il est impossible de concevoir un programme qui détecte précisément tous les programmes malveillants et (2) il est impossible de déterminer si un programme est un virus ou non car le problème est indécidable, donc les faux positifs et négatifs sont inévitables. Autrement dit, il n'existe pas de détecteur parfait et aucun d'algorithme ne permet de savoir précisément si un programme est un logiciel malveillant ou non. Nous utilisons alternativement le mot virus et logiciel malveillant ci-dessus, ceux-ci ont la même signification pour nous ici. L'analyse statique n'est donc pas retenue car complexe à mettre en œuvre en pratique (e.g., puissance de calcul, temps). De plus, elle souffre des mêmes limitations théoriques que l'analyse dynamique. Un argument supplémentaire est que

nous nous intéressons au comportement malveillant et pas au contenu du code, contrairement à l'analyse statique. Analyse statique et dynamique sont donc complémentaires, bien qu'ici la première ne soit pas retenue.

La détection doit s'orienter vers des méthodes statistiques afin de limiter : (1) les infections et (2) les fausses alertes, c'est-à-dire principalement des pertes de services pour l'utilisateur. Pour cela, il faut étudier le comportement courant des utilisateurs.

Une analyse dynamique doit se dérouler dans un environnement réaliste pour le logiciel malveillant, afin que celui-ci s'exécute normalement. Les attaquants conscients que les binaires sont activement analysés mettent au point des mécanismes de défense. En effet, ils tentent de déterminer s'ils s'exécutent dans un environnement d'analyse, dont l'activité et le contenu sont simulés par exemple. De nombreux artefacts sont utilisés pour détecter des environnements factices (e.g., déplacement souris). Les environnements d'analyse sont appelés "sandbox" ou "bac à sable" et utilisent différentes technologies. De plus, ceux-ci sont accompagnés d'outils de surveillance, c'est-à-dire d'introspection, pour observer l'effet du code sur le système. Il faut donc considérer : (1) la solution ou l'architecture plébiscité pour exécuter le code et (2) les moyens mis en oeuvre lors de son analyse. Le plus souvent, les deux éléments précédents sont liés, entendez, l'architecture détermine les moyens à disposition pour l'analyse. Le bac à sable dans son ensemble doit être le plus transparent possible, c'est-à-dire indistinguable par rapport à un environnement utilisateur classique. Bulazel *et al.* [68] répertorient l'ensemble des architectures disponibles à ce jour dans le TABLEAU 1.1. Les architectures les plus faciles à déployer, soit l'émulation et la virtualisation, sont les plus simples à évader pour les logiciels malveillants. À l'inverse, les plus coûteuses à déployer et maintenir, à savoir la virtualisation native et les machines nues, sont plus difficiles à détecter.

Tableau 1.1 – (Bulazel *et al.* [68] [5]) Une comparaison de l'architecture des différentes techniques pour l'analyse de binaires. Vision haut niveau des avantages et désavantages de chacune d'elles.

	Emulateur	**Machine virtuelle**	**Hyperviseur**	**Machine nue**
Architecture	Le code est exécuté par du logiciel	La majorité du code s'exécute sur du matériel	Virtualisation directement sur matériel	Pas de virtualisation
Transparence	Basse	Basse	Moyenne	Haute
Avantages	Contrôle total	Facile à utiliser & grand contrôle	Impact faible & proche du matériel	Pas d'artefacts liés à la virtualisation
Désavantages	Artefacts & performance	Artefacts	Arfefacts & introspection	Coût & mise à l'échelle difficile & faible introspection
Exemples	*QEMU* [66]	*VirtualBox* [41]	*Xen* [65]	*BareBox* [92]

5. reproduit avec la permission de *Association for Computing Machinery, Inc (ACM)*

La virtualisation est définie comme suit par *VMware*[6] : "*La virtualisation est le processus qui consiste à créer une version logicielle (ou virtuelle) d'une entité physique*". Nous utilisons ensuite le terme invité pour désigner le système virtualisé et le terme hôte pour le système qui réalise la virtualisation grâce au "virtual machine monitor" (VMM). Nous distinguons deux types de virtualisation : (1) virtualisation hôte, machine virtuelle et (2) virtualisation native, hyperviseur. Dans le premier cas, un système d'exploitation hôte est indispensable. Pour le second, le matériel et le VMM assurent la gestion de l'invité sans système d'exploitation hôte. Le système virtualisé doit partager la même architecture matérielle que l'hôte pour (1) et (2), car la plupart des instructions sont exécutées sur le matériel. L'émulation au contraire exécute toutes les instructions en logiciel. L'introspection est donc aisée. De plus, il n'est pas nécessaire que l'invité partage la même architecture que l'hôte. Des artefacts sont inhérents à la virtualisation [78] et l'émulation, ce qui rend les solutions basées sur ces approches facilement détectables. De nombreuses solutions clés en main existent pour chacune des approches présentées dans le TABLEAU 1.1. Nous ne discutons pas ici, des problématiques liées à la création de contenu réaliste. Nous entendons par cela, que la machine victime paraisse vraisemblable pour un attaquant. Par exemple à travers, les fichiers et répertoires présents, les applications installées, etc.

L'utilisation de machines nues comme plateforme d'analyse de logiciels de rançon possède un avantage et pas des moindres : absence totale de virtualisation ou d'émulation. Une telle plateforme est donc optimale en terme de transparence. L'introspection par contre est plus compliquée à mettre en place et à un impact sur la transparence. Nous présentons chapitre 2 notre plateforme d'analyse automatique de logiciels malveillants, celle-ci est composée de machines nues et embarque un pilote de filtre pour observer le comportement des binaires.

1.4 Conclusion

Dans ce chapitre nous avons introduit les notions essentielles à la compréhension du lecteur. Celles-ci sont centrées autour de trois axes : (1) présentation du système d'exploitation Windows, (2) fonctionnement du système de fichiers sur cette même plateforme et (3) l'analyse dynamique.

Une des contributions de cette thèse, soit une contre-mesure aux logiciels de rançon, est orientée détection temps réel. En effet, nous nous intéressons aux effets du code sur le système de fichiers lors de nos analyses et pas au code lui même. Notre contre-mesure doit donc observer un comportement malveillant avant de le bloquer, par opposition à l'analyse statique.

Dans le chapitre suivant nous présentons l'état de l'art concernant les logiciels de rançon et plus particulièrement les contre-mesures associées. Une particularité est à noter, la plupart des contributions sont très récentes et concurrentes au déroulement de cette thèse.

6. https ://www.vmware.com/fr/solutions/virtualization.html

Chapitre 2

Les logiciels de rançon : revue de littérature

Le chapitre précédent introduit des connaissances générales relatives à l'analyse des logiciels malveillants. Dans ce chapitre nous présentons une revue de littérature concernant les logiciels de rançon en nous intéressant tout spécialement aux contre-mesures proposées pour lutter contre ces logiciels. Rappelons que l'objectif de cette thèse est de mettre au point une solution avec un faible impact sur le système tout en restant efficace en terme de détection. Cet état de l'art est composé de deux parties bien distinctes. La première comprend les contributions avant 2015, peu nombreuses, en effet à cette époque les logiciels de rançon ne représentaient qu'une partie très marginale des logiciels malveillants. La seconde partie est bien plus étoffée. En effet l'année 2015, qui coïncide avec le début de cette thèse, marque le début de l'explosion du nombre d'attaques de logiciels de rançon. En conséquence, de nombreuses contributions académiques s'en sont suivies.

2.1 Du "cryptovirus" au logiciel de rançon moderne

En 1996 paraît le tout premier article qui pose les fondements théoriques d'une menace encore hypothétique à cette époque appelée : "cryptovirus". Young et Yung [112] mettent en avant une idée simple : l'utilisation de la cryptographie à des fins malveillantes. L'objectif est de prendre le contrôle des données de l'utilisateur pour obtenir une rançon. La notion de base sur laquelle repose leur modèle est l'utilisation d'un système hybride. Ce système mêle à la fois un algorithme de chiffrement symétrique et asymétrique. Un tel système est encore utilisé en 2018 par la plupart des logiciels de rançon. Leur modèle introduit également la notion de "high survivability", on peut traduire cela par la capacité d'un virus (i.e., h-s virus) à se greffer sur une machine hôte de sorte à la rendre dépendante. Autrement dit, la récupération des données ne peut se faire qu'au bon vouloir de l'auteur du virus, ce qui permet au logiciel de rançon de survivre (e.g., données cruciales). L'altération ou la suppression du h-s virus entraîne la perte des données de manière irrémédiable. Pour survivre, l'effet produit sur le système doit être irréversible, les primitives cryptographiques rendent impossible la résolution du problème en un temps raisonnable (i.e., algorithme polynomial). On pense tout particulièrement au chiffrement à clé publique, pierre

angulaire du modèle hybride puisqu'il assure la sécurité des clés secrètes.

Formalisons le modèle hybride de Young et Yung [112] :

$$\begin{aligned} RNG &= \textit{générateur de nombres aléatoires} \\ K_s &= \textit{clé de session} \\ IV &= \textit{vecteur d'initialisation} \\ K_{pub}, K_{priv} &= \textit{paire clé publique et clé privé, bi-clé} \\ m &= \{IV, K_s\} \\ m^{'} &= \{IV, K_s\}\,,\, K_{pub} \end{aligned} \tag{2.1}$$

Pour chaque machine infectée, le logiciel de rançon génère une clé de session K_s et un vecteur d'initialisation IV avec son générateur de nombres aléatoires. Cette clé de session est en fait une clé secrète utilisée par l'algorithme de chiffrement symétrique pour chiffrer les documents de l'utilisateur. Une clé publique K_{pub} est présente dans le binaire et seul l'auteur du "cryptovirus" possède la clé privée associée K_{priv}. Une bi-clé K_{pub}, K_{priv} peut être créée pour chaque cible, pour chaque campagne, etc. Une fois les documents chiffrés, la clé de session et le vecteur d'initialisation le sont également. Il s'agit d'un message clair m transformé par un chiffrement à clé publique en un chifrré $m^{'}$ grâce à K_{pub}. Les secrets sont ensuite effacés de la mémoire vive. Pour récupérer ses données l'utilisateur doit alors fournir le message $m^{'}$ ainsi que la rançon. Il obtient alors la clé de session K_s et IV. Tous deux déchiffrés par le criminel grâce à K_{priv}. La connaissance d'une clé de session ne permet pas d'aider d'autres victimes et K_{priv} n'est jamais divulguée. Les cryptosystèmes hybrides sont utilisés dans une grande variété d'applications, comme par exemple les mails sécurisés avec Pretty Good Privacy.

> *"A cryptovirus is equipped with a strong random number generator and a strong seeding procedure and mounts an attack by generating a random session key K_s and a random initialization vector IV."*

Il est intéressant de noter que pour Young et Yung [112], un logiciel de rançon est équipé de son propre RNG. Difficile de savoir si cette position s'est imposée par elle-même car au moment de la rédaction de leur papier il n'y avait pas de librairie cryptographique publique, ou s'ils ont anticipé d'éventuelles contre-mesures à venir. En effet, comme nous allons le voir dans la seconde partie, le fait d'utiliser un générateur aléatoire tiers (i.e., librairie) est une faiblesse de conception qui peut-être exploitée.

Ce modèle hybride est toujours en vigueur, à une exception près, certaines familles préfèrent générer la bi-clé directement sur la machine hôte. Dans ce cas, une communication avec un serveur est nécessaire pour exfiltrer la clé privée. Ce modèle permet d'utiliser de multiples clés de session K_s. Une fois la rançon payée, K_{priv} est divulguée, car elle n'a pas de valeur pour les autres victimes. Il n'est donc pas nécessaire de demander le déchiffrement de $m_1^{'}$, $m_2^{'}$, $m_3^{'}$, etc.

Young et Yung [112] présentent également leur implémentation d'un "cryptovirus" sur Macintosh pour démontrer la faisabilité de leur démonstration. Celui-ci ne dépasse pas 7k octets et embarque l'ensemble des fonctions cryptographiques. En 2006, Young *et al.* [111] présentent une nouvelle preuve de concept toujours basée sur le modèle hybride présenté précédemment. Celle-ci

détourne les fonctions cryptographiques d'une librairie publique afin d'infecter un utilisateur. Il s'agit de l'interface de programmation d'applications cryptographiques ou plus communément appelée CryptoAPI de Microsoft. Celle-ci fait son apparition en 1996 et est intégrée dans Windows 95. On la retrouve encore aujourd'hui sur Windows 10 pour des questions de rétrocompatibilité.

Young et Yung [112] suggèrent néanmoins des contre-mesures, la première est de contrôler les accès aux outils cryptographiques (i.e., inexistant en 1996) afin d'en identifier les usages malveillants. Ils discutent également du fait de fournir des primitives cryptographiques aux utilisateurs. Pour eux cela réduit la sécurité du système si celui-ci est susceptible de se faire infecter. Cela permet également aux criminels de créer des "cryptovirus" sans connaissance particulière en cryptographie. Les logiciels de rançon n'étant qu'un raffinement des logiciels malveillants, ils proposent de les combattre de la même façon : avec des antivirus de manière proactive. Bien entendu la sauvegarde des données est également conseillée.

Dix ans plus tard, Young *et al.* [111] introduisent deux nouvelles contre-mesures basées sur l'utilisation de la CryptoAPI. Avant de procéder au chiffrement d'un texte en clair avec une clé publique, il est nécessaire de prouver que l'on possède la clé privée associée. Ils proposent d'utiliser une preuve à divulgation nulle de connaissance pour implémenter ce mécanisme. Par exemple, lors de son authentification l'utilisateur insère une carte à puce, qui contient sa clé privée, dans la machine afin de réussir le(s) challenge(s) du vérifieur. Ils proposent également l'utilisation de certificats provenant d'autorités de confiance. Si la clé publique provient d'un tel certificat alors on peut raisonnablement penser que son interlocuteur n'est pas mal intentionné. Dans les deux cas, un pilote est incorporé (i.e., "cryptographic verifier") dans le noyau du système d'exploitation afin de vérifier : (1) l'utilisateur réussit la preuve à divulgation nulle de connaissance, (2) la signature du certificat. En se positionnant ainsi dans le noyau on souhaite évoluer dans un contexte différent des logiciels de rançon, la solution est transparente et l'on dispose de davantage de privilèges. En revanche, aucune solution n'est proposée pour stopper un logiciel de rançon qui embarquerait l'ensemble des primitives cryptographiques directement dans son binaire. De plus, il semble compliqué dans le contexte actuel, où la cryptographie est omniprésente de mettre en place les solutions (1) et (2) proposées ci-dessus. En effet, un certificat signé par une autorité de confiance n'est pas nécessaire dans certains cas d'usages, de même que posséder la clé privée.

En 2008, Gazet [79] procède à une analyse comparative des logiciels de rançon les plus répandus. Il désassemble ainsi 15 logiciels de rançon dont 8 provenant de la famille *Gpcode*. Il met en exergue le manque de connaissance en cryptographie de leurs auteurs, en effet tous les échantillons présentent des faiblesses de conception. On trouve néanmoins des cryptosystèmes hybrides comme présenté en 2.1, mais uniquement dans les échantillons de la famille *Gpcode*. Des vulnérabilités sont malgré tout présentes : taille du module de l'algorithme RSA trop faible, générateur de nombres pseudo-aléatoires non sûr ou recherche exhaustive de la graine possible, etc. À noter qu'aucun des échantillons n'utilisent la CryptoAPI, ou du moins aucune mention n'en est faite. Ils embarquent des librairies tierces ou possèdent leur propre implémentation. Les derniers échantillons de *Gpcode* (i.e., version *ai*), qu'on peut qualifier de logiciel de rançon moderne, présentent les capacités les plus avancées : injection de thread, crochet de fonction, protocole de communication, etc. Gazet met en avant le fait que ce n'est pas dans l'intérêt des

auteurs de logiciels de rançon de mettre au point un cryptosystème parfait. Cela risquerait d'attirer l'attention des autorités ainsi que des antivirus. En 2008 le mot d'ordre est donc pour Gazet : *"Few investments, few incomes, few risks."* Les entreprises ne sont pas encore ciblées, seulement les particuliers, qui possèdent une capacité de rétorsion très faible. De plus, le nombre d'infections par logiciels de rançon est encore très marginal. Le blanchiment de l'argent est évoqué, il s'agit d'un problème pratique de taille. Gazet prédit même que les logiciels de rançon sont certainement voués à l'échec. Il faudra attendre la démocratisation du Bitcoin quelques années après pour faire exploser le nombre d'infections par des logiciels de rançon.

Dans cette première partie de la revue de littérature, on passe d'une menace hypothétique en 1996 avec le "cryptovirus" à des réalisations bien réelles mais encore imparfaites en 2008 de logiciels de rançon modernes. Une question reste en suspens : Comment bloquer un logiciel de rançon qui embarque ses propres primitives cryptographiques ? En effet, les travaux avaient pour objectifs de démontrer la faisabilité d'un logiciel de rançon avec les propriétés de Young et Yung [112]. Ce qui explique qu'aucune contre-mesure n'ait encore été implémentée et testée, et ce même concernant la CryptoAPI. La seconde partie va combler se vide devant l'urgence. En effet, à partir de 2013 les logiciels de rançon vont devenir très virulents.

2.2 Une pléiade de contre-mesures

Le nombre d'attaques par logiciels de rançon passe de quasi-inexistant en 2012 à plus de 15k en 2013 pour un seul éditeur d'antivirus, Kaspersky. Son rapport 2018 [50] fait état d'au moins 1.5 million et 950k attaques pour les années 2016 et 2017. Le monde académique va donc s'intéresser de nouveau à cette menace. Cette section présente des contre-mesures pour le système d'exploitation Windows. En effet, il compte à lui seul pour 70% des logiciels malveillants détectés dans le monde [48]. Les preuves de concept sont donc réalisées sur ce système d'exploitation, il n'en demeure pas moins que les solutions présentées sont transposables à d'autre systèmes. Des contre-mesures vont être proposées, deux paradigmes s'opposent :

— contrôle des accès aux primitives cryptographiques, sous-section 2.2.1,

— détection comportementale via les accès aux supports de stockage, sous-section 2.2.2.

Cette section va être partagée selon ces deux points de vues.

2.2.1 Contrôle des accès à la CryptoAPI

Les logiciels de rançon ont besoin de cryptographie "forte", c'est-à-dire d'utiliser des primitives sûres et correctement implémentées. La CryptoAPI est une des premières librairies publiques offrant de tels services, sortie en 1996 sur Windows 95. À titre de comparaison, OpenSSL ne voit le jour qu'en 1998. La CryptoAPI permet d'accéder à une source d'entropie avec un PRNG, exporter et/ou importer des secrets, de disposer d'algorithme de chiffrement symétrique et asymétrique, de calculer des sommes de contrôle, etc [6]. L'ensemble de ces primitives sont implémentées dans des "cryptographic service providers" ou fournisseurs de service cryptographique de différents types.

Par défaut, les fournisseurs de Microsoft sont présents sur un système, mais rien n'empêche un utilisateur ou une entreprise d'ajouter son propre fournisseur [1] pour davantage de contrôle (e.g., la gestion des clés).

Ransomware and the Legacy Crypto API [PBL+16]

Nous avons été les premiers dans *Ransomware and the Legacy Crypto API* [PBL+16] en 2016 à présenter une contre-mesure et son implémentation basée sur la CryptoAPI. L'idée étant de contrôler les accès aux primitives cryptographiques via l'implémentation d'un "cryptographic service provider". Nous avons intégré notre propre fournisseur de services cryptographiques sur une machine de test. Nous forçons ensuite son emploi au détriment : (1) du désir de l'utilisateur et sinon (2) du fournisseur par défaut. Nous reviendrons plus en détail dessus dans la section 4.

PayBreak : Defense Against Cryptographic Ransomware [93]

En 2017, Kolodenker *et al.* [93] présentent une solution appelée PayBreak. Il s'agit d'une solution de remédiation post-infection. Les auteurs font l'hypothèse que les logiciels de rançon utilisent des primitives cryptographiques sûres et éprouvées. Leur modèle d'un attaquant prend donc en considération l'usage de la CryptoAPI mais aussi de librairies statiques. En effet comme nous l'a montré Gazet [79] le développement de leurs propres algorithmes mène inévitablement à des erreurs. L'usage de la CryptoAPI et de librairies standards s'est donc répandu. PayBreak est constitué de trois composants, tous trois situés dans l'espace utilisateur :

1. crochet de fonction,
2. coffre-fort
3. restauration des fichiers.

Les crochets de fonction ont pour objectif de détourner le flot d'exécution d'un programme, comme présenté en 1.1.3. L'objectif est de copier les arguments de plusieurs fonctions "critiques" d'un point de vue cryptographique. Les crochets de fonction sont réalisés avec la librairie *Detours* [85] de Microsoft Research. Concernant la CryptoAPI, plusieurs fonctions sont placées sous surveillance : *CryptEncrypt*, *CryptGenRandom*, *CryptSetKeyParam*, etc. Seule la librairie Crypto++ est considérée pour une édition de lien statique et des fonctions similaires à celles de la CryptoAPI sont surveillées. Les auteurs souhaitent ainsi récupérer les clés symétriques, les vecteurs d'initialisations mais aussi les algorithmes de chiffrement utilisés et leurs modes de chaînage. PayBreak récupère également les données provenant des PRNGs. Il s'agit d'une précaution en cas d'attaque pour aider post-mortem des analystes.

Pour crocheter une fonction, il est nécessaire de connaître son adresse. Un exécutable qui utilise des fonctions provenant de librairies dynamiques (e.g., CryptoAPI) contient leurs adresses dans son Import Address Table. Avec une édition de lien statique, il faut faire la recherche manuellement, par exemple avec une reconnaissance par signature comme c'est le cas ici. En effet, l'adresse des fonctions ciblées dans Crypto++ n'apparaît pas dans l'IAT, le code est présent

1. Pour fonctionner correctement un fournisseur de services cryptographiques doit être signé par Microsoft

dans le binaire. La reconnaissance par signature se fait en calculant un score de similarité (i.e., fuzzy hash) sur les 32 premiers octets d'une fonction.

D'un point de vue pratique les crochets sont réalisés en injectant en temps réel une librairie dynamique (i.e., la charge utile) qui utilise l'API de *Detours* dans l'espace mémoire d'un processus. Les crochets vont ensuite être créés automatiquement dans les fonctions spécifiées, sans que l'utilisateur n'ait à se soucier des considérations techniques (i.e., modification d'instructions). L'injection est exécutée selon la méthode *AppInit_DLLs*. Tous les processus du système qui utilisent *user32.dll* vont charger la charge utile de PayBreak. Une limitation demeure si le binaire malveillant n'utilise pas *user32.dll*, alors les crochets ne sont pas appliqués. De plus, il semble d'après le code source [2] de PayBreak que les auteurs n'ont pas pris en compte l'emploi de la CryptoAPI via les fonctions *LoadLibrary* puis *GetProcAddress*, l'efficacité de leur implémentation est par conséquent imparfaite.

Les crochets sur Crypto++, c'est-à-dire pour une édition de lien statique sont plus compliqués à mettre en place. Il est nécessaire pour la charge utile d'inspecter la mémoire du processus dans lequel elle s'exécute. Pour cela, les auteurs de *PayBreak* scannent la mémoire après le premier appel de *NtReadFile*. Il est plus probable de trouver des opérations de chiffrement après des accès disque. Une fois l'adresse d'une fonction connue trouvée grâce à sa signature, les auteurs réalisent un crochet. La recherche par signature est sensible à l'offuscation, en effet, les auteurs ne considèrent pas que les logiciels de rançon puissent utiliser une telle technique. Ils prennent seulement en considération le paquetage, mais celui-ci n'a aucun impact sur la reconnaissance par signature. Ce qui est paradoxal car le paquetage est souvent accompagné par des techniques d'offuscation.

Le second composant de PayBreak est une autorité de séquestre, les données critiques provenant du composant (1) sont stockées dans un coffre-fort (i.e., key vault) chiffré. Celui-ci est chiffré avec une clé publique. Les premières étapes avant de déployer PayBreak sur une machine sont : (A) générer une bi-clé (B) importer la clé publique propre à cette instance. Pour des raisons de sécurité la clé privée est stockée sur une autre machine ou un support déconnecté de tout réseau. Ce coffre-fort n'est accessible qu'avec les droits administrateurs et on ne peut obtenir un descripteur de fichier qu'en ajout uniquement. Les auteurs conscients des limites d'une autorité de séquestre, (e.g., abus), insistent sur le fait qu'ici l'utilisateur possède l'accès exclusif à ses clés. La gestion de cette autorité par un processus s'exécutant dans le noyau améliorerait la sécurité de la solution.

Le dernier composant est responsable de la restauration des fichiers. Une fois l'infection constatée, il est nécessaire d'utiliser la clé privé, jusque-là gardée en sécurité (e.g., support externe) pour accéder aux informations du coffre-fort. PayBreak va ensuite tester de manière itérative toutes les clés symétriques du coffre-fort avec leurs vecteurs d'initialisation et le mode de chaînage correspondant sur un chiffré. Il est aussi nécessaire de faire une recherche exhaustive sur le décalage introduit par le logiciel de rançon avec l'ajout de ses métadonnées. PayBreak arrête la recherche quand un en-tête de fichier connu est détecté par la librairie *libmagic*, celle-ci identifie les types de fichiers. Une fois que PayBreak est en possession du décalage et de la

2. https://github.com/BUseclab/paybreak

clé, celui-ci procède au déchiffrement complet du système. Le logiciel de rançon peut également utiliser une clé par fichier ou type de fichier, dans ce cas, le déchiffrement est plus long. Les auteurs discutent de la possibilité de recourir à la programmation dynamique pour optimiser la recherche des paramètres. L'approche exhaustive étant suffisante pour cette preuve de concept. La FIGURE 2.1 présente l'architecture, les trois composants et le fonctionnement de PAYBREAK lors d'une attaque.

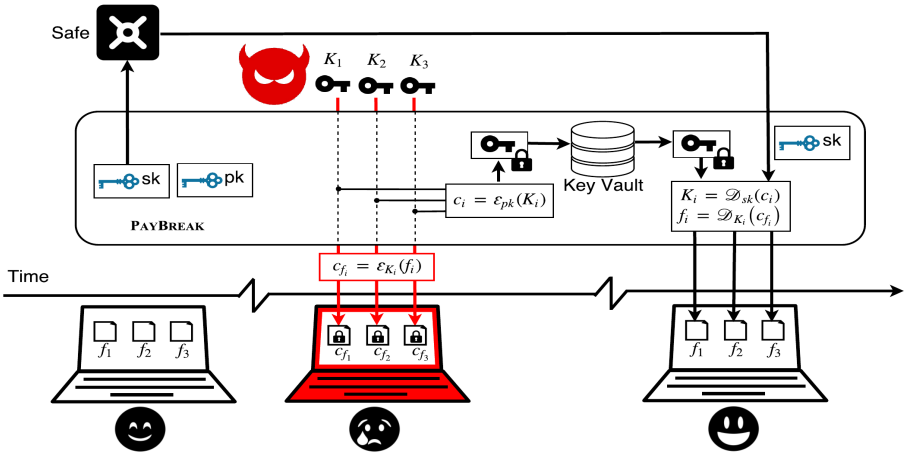

FIGURE 2.1 – (Kolodenker *et al.* [93][3]) Architecture de la contre-mesure PAYBREAK.

Les expérimentations se déroulent avec un outil développé par Kolodenker *et al.* : *RADDAR* [35]. Son objectif est d'automatiser la récupération de binaire depuis des dépôts en ligne puis de les placer dans la *Cuckoo Sandbox* [10]. Il s'agit d'une plateforme d'analyse automatique de logiciels malveillants open-source basée sur la virtualisation (e.g., KVM, VirtualBox). Cette plateforme dispose de nombreuses fonctionnalités pour aider un analyste : image mémoire, surveillance des appels systèmes, trafic réseau, etc. La première étape est de récupérer des échantillons valides. Pour cela, les auteurs les exécutent 20 minutes puis regardent s'ils ont exhibé un comportement malveillant. Sur un total de 703 échantillons récupérés avec un label de logiciel de rançon dans différentes bases de données, seul 107 appartenant à 20 familles sont actifs dans la *Cuckoo Sandbox* au moment des tests. Les raisons invoquées, sont principalement le "fingerprinting" de l'environnement, en effet, les techniques anti-émulation détectent facilement un environnement virtualisé qui est bien distinct d'une machine "bare-metal".

Pour les résultats, 12 familles sur les 20 sont déjouées avec succès, on se sait pas à quelle proportion des échantillons cela correspond. Les logiciels de rançon qui utilisent la CryptoAPI sont contrecarrés sans surprise, une famille utilisant Crypto++ également. Les faux négatifs sont principalement, c'est-à-dire 5/8, des familles qui utilisent leurs propres algorithmes cryptographiques. Les auteurs ajoutent que pour ces cinq familles, des problèmes de conception permettent de ne pas payer la rançon. Pour les trois dernières familles on ne sait pas quelle librairie est utilisée. Dans ce cas, le composant (1) est inefficace et aucune clé n'est consignée dans le coffre-fort. Le composant (3), en charge de la restauration a été capable de déchiffrer l'ensemble des documents

3. Reproduit avec la permission de *Association for Computing Machinery, Inc (ACM)*

utilisateur pour les échantillons des 12 familles déjouées. Il faut ainsi 360 minutes pour déchiffrer 10k fichiers chiffrés par *Locky*. Les auteurs évaluent la restauration d'un système complet à plusieurs heures.

Le coût pour le système d'une telle solution est faible, aucune latence n'est perceptible pour un utilisateur. Pour 10 millions d'appels à *CryptEncrypt* on constate un surcoût de 124 μs en moyenne par appel. Il faut donc une seconde de plus pour compléter ce test de performance avec PayBreak ce qui est négligeable. Le composant (1) scanne la mémoire des processus à la recherche de fonctions appartenant à des librairies statiques, cette recherche peut avoir un coût sur les performances du processus, les auteurs n'en discutent pas.

D'un point de vue des limitations, la signature statique n'est pas résiliente à l'utilisation de différents compilateurs et à l'offuscation. Il faudrait donc une signature par compilateur. De plus, il paraît compliqué de couvrir l'ensemble des librairies cryptographiques existantes ainsi que l'ensemble des compilateurs. D'autre part, en ayant connaissance du fonctionnement de PayBreak, le coffre-fort peut être corrompu en injectant de nombreuses clés. L'objectif est ainsi de ralentir la récupération des données. Une heuristique pour mettre en place une rotation des données dans le coffre-fort est aussi à envisager afin de prévenir un accroissement illimité de sa taille. Il est aussi possible pour les logiciels malveillants de détecter la mise en place des "trampolines" de *Detours* pour réaliser les crochets, cette technique étant très répandue dans les outils de sécurité.

PayBreak a pour avantage d'adresser le problème des performances et de permettre un déploiement facile. Malheureusement il s'agit d'une solution de remédiation post-infection. Il semble donc ne pas y avoir de grandes différences entre un système de sauvegarde correctement configuré et PayBreak. Les auteurs se basent sur des statistiques provenant d'une firme de sécurité [54], pour indiquer qu'une solution complémentaire aux sauvegardes est nécessaire. En effet, celles-ci sont déficientes de manière significative pour une variété de raisons : migration, mise à jour, sauvegarde non testée, etc.

Ransomware-Prevention Technique Using Key Backup [94, 95]

En 2017, Lee *et al.* [94, 95] présentent indépendamment une contre-mesure similaire à PayBreak mais qui s'appuie sur Cryptography API: Next Generation. La librairie CNG est la remplaçante de la CryptoAPI. Elle dispose de primitives cryptographiques plus récentes. La contre-mesure est une autorité de séquestre. Le but est de conserver les clés secrètes et les bi-clés générées sur une machine afin de permettre la restauration des fichiers si une attaque est détectée (e.g., note de rançon). Il s'agit d'une solution de remédiation post-infection. Les auteurs mettent en place des crochets sur l'API de CNG à différents emplacements : chiffrement, import et génération de clé. Le crochet sur la fonction *BCryptEncrypt* permet de récupérer les paramètres du chiffrement (e.g. vecteur d'initialisation) mais aussi la clé secrète dans l'hypothèse où celle-ci n'est pas générée par CNG. Les expérimentations s'appuient sur deux programmes développés par les auteurs : (1) un logiciel de rançon "home-made" utilisant CNG et (2) le programme de protection. Aucune expérimentation n'a lieu sur des logiciels de rançon réels. En effet, il n'a pas encore été reporté l'utilisation de la librairie CNG par un seul logiciel de rançon d'après mes

connaissances en 2018. Si tel était le cas, son utilisation serait très marginale. Le programme de protection est activé manuellement par l'utilisateur pour restaurer les fichiers. Les auteurs pour palier ce problème proposent d'ajouter de l'apprentissage automatique. De nombreuses parties ne sont pas détaillées : implémentation du programme de protection, limitations de l'approche, restauration des fichiers, etc.

No Random, No Ransom : A Key to Stop Cryptographic Ransomware [81]

En 2018, Genç *et al.* [81] présentent une nouvelle contre-mesure, basée sur une idée simple. Les logiciels de rançon se doivent d'utiliser une source d'aléa de la meilleur qualité possible car la sécurité des clés en dépend. Pour cela, ils font appel à la CryptoAPI et à sa fonction : *CryptGenRandom*. En effet, c'est via cette fonction que sont générées les clés secrétes, les PRNGs disponibles sur un système (i.e., Windows) étant limités. Le contrôle des applications qui accèdent à *CryptGenRandom* permettrait de stopper en temps réel une attaque. Les auteurs indiquent également que ça n'est pas dans l'intérêt des criminels d'utiliser un PRNG d'une librairie tierce ou d'implémenter leur propre générateur pseudo-aléatoire. Dans le premier cas, le problème de son initialisation avec le choix de la graîne est posé : comment obtenir une graîne non déterministe ? Des failles de sécurité de ce type étant apparues dans Debian [45] et des produits Jupiter [47], en 2008 et 2015. Le résultat est une sortie des générateurs prédictibles. Pour le second cas, comme nous l'a montré Gazet [79] les implémentations "home-made" sont facile à cryptanalyser, tout du moins en 2008. Cette contre-mesure appelée USHALLNOTPASS est donc la première basée sur la CryptoAPI capable de détecter en temps réel une attaque avant que les fichiers de l'utilisateur ne soient touchés.

Avec un tel modèle de l'attaquant, USHALLNOTPASS cible les logiciels de rançon les plus évolués, c'est-à-dire ceux qui utilisent un bon PRNG. Pour mettre en place le système de contrôle d'accès en fonction de l'appelant, il est nécessaire d'intercepter les appels à *CryptGenRandom*. Pour cela, les auteurs utilisent la librairie *Detours*. D'un point de vue technique l'injection de la charge utile de USHALLNOTPASS se fait de la même manière que PayBreak. À noter une différence majeure entre ces deux contres-mesure : USHALLNOTPASS ne sauvegarde pas la sortie des sources d'entropie.

Afin de n'autoriser que des processus légitimes à accéder au PRNG, les auteurs proposent d'utiliser une liste blanche. L'initialisation de cette liste blanche est donc cruciale, afin de limiter les faux positifs mais surtout les faux négatifs. La liste blanche est construite de manière empirique en observant les applications qui accèdent au PRNG sur un système supposé sain. De plus, la liste des applications peut varier d'une version à une autre d'un système d'exploitation. Les auteurs découvrent ainsi qu'une vingtaine d'applications du système sur Windows 7 accèdent au PRNG et seulement deux sont signées. Aucune solution n'est proposée pour automatiser la création de la liste blanche initiale. Conscient des limites d'une liste statique figée dans le temps, les auteurs suggèrent deux mécanismes dynamiques pour ajouter une application : contrôle humain et signature par un tiers de confiance.

Un module d'interception (INT) capte tous les appels à la fonction *CryptGenRandom*. Celui-ci transmet les informations sur le processus appelant au module de contrôle (CTR) qui consulte

alors la liste blanche pour vérifier la présence ou non de l'application appelante dans celle-ci. Pour cela, Genç *et al.* comparent à l'aide d'une fonction de hachage cryptographique (i.e., SHA-256) le haché correspondant à l'image de l'application appelante sur le disque aux hachés présents dans la liste blanche. La liste blanche est en fait une liste de hachés présents sur le disque que seul l'administrateur peut modifier. L'intégrité de la liste blanche est assurée par un code d'authentification de message. Si l'application n'est pas autorisée, le module de contrôle termine l'application avec la fonction *ExitProcess*. Ce contrôleur dispose d'une interface graphique permettant de voir en temps réel les accès à *CryptGenRandom* mais aussi d'ajouter et de supprimer des applications de la liste blanche.

Les expérimentations sont menées avec la *Cuckoo Sandbox*. Une image d'un Windows 7 sain avec des fichiers témoins est utilisée. Chaque analyse dure 20 minutes, puis vérifie ensuite l'intégrité des fichiers témoins. Les échantillons sont récupérés depuis des dépôts en ligne. Sur 2263 échantillons portant des labels de logiciels de rançon, 524 sont bel et bien des logiciels de rançon encore actifs. La vaste majorité des échantillons n'a pas exhibé son comportant malveillant pour deux raisons : environnement d'analyse et serveur distant éteint. Les résultats sont excellents, l'ensemble des 31 familles des 524 échantillons sont bloqués. Ce qui correspondant par contre à un taux de détection de 94%. Les 6% restant ne sont pas détectés à cause de problèmes techniques liés à l'implémentation du module d'interception (INT) de USHALLNOTPASS. Les échantillons non détectés font bien appel à *CryptGenRandom* d'après une "analyse dynamique". Malheureusement ils n'expliquent pas comment celle-ci a été réalisée. Les résultats de l'expérimentation montre que les logiciels de rançon font de nombreux appels à la fonction *CryptGenRandom* (i.e., plus de 20k pour 55 échantillons de *Locky*).

Cette solution adresse également le problème des performances. Celle-ci de part sa nature, un crochet sur une seule fonction est très minimaliste. Nous nous attendons à d'excellent résultats. Les auteurs réalisent des tests de performances en évaluant l'impact sur la CryptoAPI et des applications courantes. Ainsi pour 100k appels à *CryptGenRandom*, Genç *et al.* constatent une augmentation avec un facteur de 125 entre le système nu et le système avec USHALLNOTPASS. Les auteurs passent de $0.12s$ pour compléter le test à $15.59s$. Les auteurs indiquent que ces médiocres performances sont dues au canal de communication entre le module d'interception et le module de contrôle qui consomment $14.9s$ du test précédent. Des améliorations techniques permettraient d'améliorer les performances. Une multitude d'applications préalablement ajouté à la liste blanche est également essayé (e.g., Skype, DropBox). Les auteurs ne remarquent aucun ralentissement, aucun problème d'instabilité et pas de perte de fonctionnalités. La latence introduite par USHALLNOTPASS est donc imperceptible pour un utilisateur.

Certaines questions ne sont pas abordées, notamment concernant les moyens d'exploiter les faiblesses de USHALLNOTPASS. Le contrôle d'accès est réalisé avec la granularité d'un processus. Un logiciel de rançon peut injecter sa charge utile en lançant un fil d'exécution dans un processus légitime. Dans ce cas, l'accès au PRNG est autorisé car *GetModuleFileName* retourne le chemin vers le binaire du processus sain appartenant à la liste blanche. De plus, il est très simple de viser un processus sain. En effet, Windows pour fonctionner normalement à besoin d'un nombre conséquent de processus systèmes (e.g., *explorer.exe*, *lsass.exe*, etc.) et ceux-ci sont bien connus. De manière similaire, un logiciel de rançon peut exploiter les fonctionnalités des

applications *rundll32.exe* et *svchost.exe*. Ces deux applications permettent de démarrer des services présents dans des librairies. Il s'agit d'hôtes bien commodes pour lancer la librairie de son choix. Celle-ci apparaît ensuite dans la liste des processus et *GetModuleFileName* comme une instance de *rundll32.exe* par exemple. À noter que *svchost.exe* fait partie de la liste blanche des auteurs, nous pouvons donc exploiter ce mécanisme. Il s'agit là d'une limitation très importante non prise en compte lors de la conception de cette contre-mesure.

Comme pour PayBreak, l'implémentation du crochet de USHALLNOTPASS ne prend pas en compte les appels à la CryptoAPI si l'édition de lien à lieu lors de l'exécution (i.e., run-time linking). La gestion de la liste blanche est aussi problématique, aucune stratégie n'est proposée lors de sa création initiale. Les mises à jours logicielles sont aussi un problème, il est nécessaire de modifier la liste blanche à chaque fois. Le risque d'une telle contre-mesure est d'ajouter un maximum d'applications à la liste blanche pour éviter la perte de fonctionnalité. Les deux mécanismes proposés pour ajouter dynamiquement une application, contrôle humain et signature du binaire, peuvent être abusés comme les auteurs le rapportent eux-même. Enfin, les logiciels de rançon peuvent se tourner vers une source d'entropie alternative, on peut citer : (1) serveur distant, (2) PRNG d'une librairie tierce ou "home-made". Le premier cas semble contraignant pour les auteurs de logiciels de rançon, en raison des filtrages réseaux la charge utile risque de ne pas se déclencher. Dans le second cas, il est très facile d'utiliser un PRNG sûr, le choix de la graine est plus compliqué.

En résumé, cette solution minimaliste est très efficace, 94% de taux de détection. De plus, cette contre-mesure qui se place dans l'espace utilisateur peut être incorporée dans un antivirus ou un pilote noyau très facilement. Les expérimentations nous permettent d'apprendre que la plupart des familles de logiciels de rançon font appel à *CryptGenRandom* de manière intensive. Il s'agit d'une solution temps réel qui bloque un comportement potentiellement malveillant (i.e., pré-infection), c'est-à-dire un appel à *CryptGenRandom* non autorisé. Le déploiement de cette solution du point de vue des performances est envisageable. Malheureusement USHALLNOTPASS ne traite pas certaines parties, notamment la gestion de la liste blanche. Cette solution a davantage été orientée pour adresser le problème des faux négatifs que des faux positifs.

Synthèse

Cette sous-section présente des contre-mesures qui s'appuient principalement sur une observation : l'utilisation de la CryptoAPI par les logiciels de rançon. Une limitation essentielle et pas des moindres est la présence du logiciel de rançon et de la contre-mesure dans le même environnement avec les mêmes privilèges : l'espace utilisateur. Un logiciel de rançon pourrait désactiver lui-même la contre-mesure. De plus, pour ne pas se faire détecter, il suffit à un logiciel de rançon d'embarquer les primitives cryptographiques (i.e., réimplémentation, librairie tierce).

Le tout premier article traitant des logiciels de rançon de Young et Yung [112] présente une preuve de concept d'un logiciel de rançon qui est : (1) **indétectable** par Genç *et al.* [81], (2) **empêche la remédiation** de Kolodenker *et al.* [93]. L'inefficacité de ces deux contre-mesures bien que toutes deux développées plus de 20 ans après est bien la preuve que l'approche proposée ici n'est pas satisfaisante d'un point de vue théorique.

2.2.2 Détection comportementale sur les entrées-sorties du système de fichiers

Cette sous-section présente une approche radicalement différente de la sous-section 2.2.1. Celle-ci a pour avantage d'être agnostique quant à la nature des primitives cryptographiques utilisées mais aussi d'évoluer dans un contexte différent. Cette approche pour contrer les logiciels de rançon est basée sur la surveillance en temps réel des entrées-sorties sur les systèmes de fichiers. Nous ne nous intéressons plus au code utilisé par un adversaire malveillant mais à son effet sur le système (i.e., chiffrement des fichiers). Nous nous plaçons également dans un contexte différent. Les contre-mesures sont positionnées dans le noyau pour davantage de transparence mais aussi pour être plus difficile voire impossible à détecter et désactiver. La mise au point d'un pilote dans le noyau entraîne des difficultés au niveau de l'implémentation, il est nécessaire de faire des compromis. Un des défis à relever étant d'avoir un impact le plus faible possible sur le système pour rendre la solution utilisable.

Cutting the Gordian Knot : A Look Under the Hood of Ransomware Attacks [89]

En 2015 Kharraz *et al.* [89] présentent les résultats d'une étude menée avec 1359 logiciels de rançon répartis en 15 familles entre 2006 et 2014. Il s'agit de la première étude significative depuis Gazet [79] en 2008. Cette étude poursuit deux objectifs : (1) démontrer que contrairement aux discussions dans la communauté à cette époque (e.g., [46]) il est possible de détecter et stopper les logiciels de rançon qui utilisent des techniques évoluées, (2) proposer une méthode efficace basée sur des observations scientifiques et non ad hoc. Le premier point semble ambitieux, à ce jour en 2018, les logiciels de rançon font toujours de nombreuses victimes [62].

Cette étude contient que 73 échantillons, appartenant à 4 familles qui chiffrent effectivement les données de l'utilisateur, soit à peine 5% du corpus. Les échantillons restant se contentent de bloquer l'écran ou utilisent des techniques superficielles. L'essentiel des échantillons ne possèdent donc pas les capacités nécessaires pour prendre en otage les données d'un utilisateur. Il s'agit d'une fenêtre représentative de ce à quoi ressemblent encore les logiciels de rançon en 2014.

Afin de discriminer un comportement malveillant d'un comportement sain, les auteurs souhaitent observer l'activité du système de fichier lors d'une attaque. Pour se faire, ils implémentent un pilote de type "file system minifilter driver" [52]. Le monde académique passe ainsi d'une approche orientée primitives cryptographiques (e.g., génération des clés) dans la sous-section 2.2.1 à une approche système bas niveau. Un pilote de type "file system minifilter driver" permet d'intercepter des I/Os provenant de l'espace utilisateur. Les auteurs peuvent ainsi modifier, annuler ou contrôler les informations qui sont envoyées au disque mais aussi les informations retournées aux applications utilisateur (i.e., code de retour). La surveillance se fait en spécifiant de manière explicite les I/Os à intercepter ainsi que les fonctions auxquelles les auteurs transférent le contrôle dans le pilote (i.e., "callback routine"). Ici les auteurs interceptent les requêtes d'écriture, de lecture et de changement d'attribut. Le pilote collecte pour chaque I/O les informations suivante :

$$< PName,\ PID,\ PPID,\ PreOpTime,\ PostOpTime,\ IRPFlag,\ Args,\ Result > . \quad (2.2)$$

Les auteurs obtiennent ainsi une vision très précise du comportement des logiciels de rançon vis-à-vis des fichiers de l'utilisateur. L'objectif est donc de caractériser les entrées-sorties sur le disque lors d'une attaque. Les expérimentations sont menées dans un environnement contrôlé, une fois de plus avec la *Cuckoo Sandbox* [10]. L'analyse de chaque échantillon dure 45 minutes et l'image système utilisée est un Windows XP. Les résultats sont concluants : les logiciels de rançon ciblent de manière bien particulière les fichiers de l'utilisateur. Deux stratégies sont découvertes : (A) chiffrement en place, (B) création de la version chiffrée en parcourant en lecture le clair puis suppression du clair. Il est donc possible de détecter différentes familles pour les auteurs. À noter qu'un échantillon utilise l'API de défragmentation pour accéder aux fichiers, ce qui est peu courant. Il n'empêche que la modification des fichiers est toujours visible depuis le pilote. Environ 35% des 1359 échantillons procèdent à la suppression des données si l'utilisateur ne paie pas. Il faut donc proposer une stratégie qui prend en compte : (I) chiffrement et (II) destruction.

Une nouvelle idée est donc proposée : surveiller les changements d'une structure particulière du système de fichier New Technology File System. Cette structure appellée Master File Table répertorie tous les fichiers d'un disque formaté en NTFS. Chaque fichier contient une entrée dans la table MFT qui donne des informations très précieuses via des attributs. Par exemple, la position exacte des données sur le disque avec *$DATA*. Le système de fichiers NTFS a été développé par Microsoft pour ses systèmes d'exploitation [16]. Lors d'une attaque de nombreuses entrées de la MFT sont modifiées dans un laps de temps très court. La suppression d'un fichier entraîne le changement de statut de son attribut *$BITMAP*. Celui-ci gère les informations quant au statut des unités logiques du disque (i.e., Logical Cluster Number), c'est-à-dire utilisé ou libre. Le système de fichiers est ensuite notifié que l'entrée associée à ce fichier dans la MFT peut être réutilisée ainsi que l'espace occupé par ses données. Le contenu des fichiers peut être : (i) résident dans la MFT ou (ii) disséminé à différent emplacement du disque lorsque celui-ci est de taille plus importante. Lors d'une suppression, les données qu'elles soient présentent en (i) ou (ii) ne sont pas effacées immédiatement. Dans l'attente d'une réallocation, l'entrée du fichier dans la MFT ainsi que les données ne sont pas effacées. Les auteurs veulent exploiter ce mécanisme pour récupérer les fichiers détruits.

Les auteurs proposent d'utiliser de l'apprentissage automatique supervisé pour détecter un comportement anormal du système de fichiers. Pour cela, le nombre de fichiers créés, supprimés, d'attributs modifiés et le contenu des fichiers (e.g., champs *$DATA*) sont à prendre en compte. Par exemple, si l'on observe que de nombreux fichiers contiennent dans leur attribut *$DATA* des données chiffrées et que ceux-ci sont situés dans des répertoires différents alors on peut suspecter une malveillance. En effet, il est peu probable que l'utilisateur chiffre en un temps très court des fichiers dans des répertoires différents. L'implémentation d'une telle contre-mesure peut-être réalisée avec un pilote de deux manières distinctes en surveillant : (a) la MFT ou (b) les I/Os de l'espace utilisateur. En s'intéressant à la MFT on peut espérer récupérer les fichiers détruits. Malheureusement il n'existe pas de suite de développement ni d'exemples. La seconde solution quant à elle dispose d'une documentation pléthorique et d'exemples. Elle peut être implémentée avec un pilote de type "file system minifilter". L'option (a) est privilégiée par les auteurs, c'est

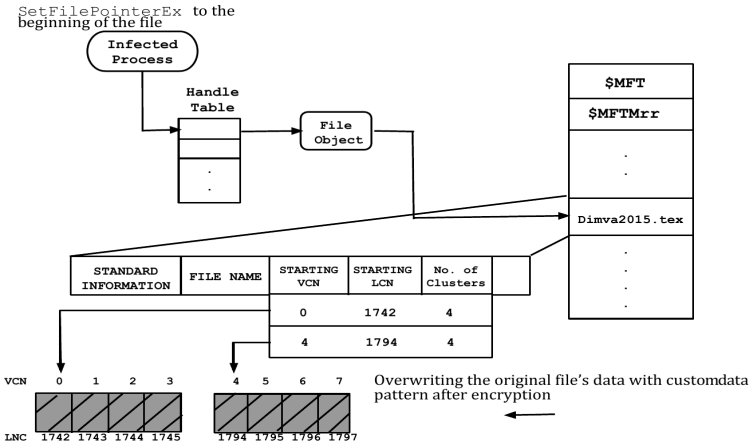

FIGURE 2.2 – (Kharraz *et al.* [89][4]) Un processus malicieux accède au contenu d'un fichier via la Master File Table (MFT) pour le chiffrer.

pourquoi le paragraphe précédent détaille quelque peu la conception du système de fichiers NTFS.

La FIGURE 2.2 présente le cheminement d'un processus malicieux pour accèder à un fichier via la MFT. Les auteurs proposent également de recourir à la surveillance des API "critiques". La troisième stratégie pour annihiler l'effet des logiciels de rançon est d'utiliser des fichiers de leurres. En effet, l'utilisation d'un modèle d'apprentissage pour caractériser le comportement courant d'un utilisateur pourrait être simulé par un programme malveillant intelligent (i.e., attaque par mimétisme). Par exemple, en s'attaquant aux fichiers les plus récemment touchés par l'utilisateur. Pour ajouter un niveau de protection à la première stratégie (i.e., surveillance MFT ou I/Os utilisateur) il est préférable d'ajouter des fichiers de leurres à de multiples emplacements du système de fichier. Il ne reste ensuite qu'à surveiller les fichiers de leurres en s'assurant qu'aucune application n'y touche. Ils sont générés de telle sorte qu'ils doivent être indiscernables pour un attaquant des fichiers classiques de l'utilisateur. De plus cette stratégie permet de détecter un logiciel de rançon, en fonction de l'emplacement des leurres, plus tôt dans son processus de chiffrement.

Cette publication ne propose pas d'implémentation mais apporte de nouvelle idées très précieuses pour stopper et bloquer les logiciels de rançon. En ce sens, l'objectif (1) formulé par Kharraz *et al.* [89] est réussi. Mais les contre-mesures proposées ne sont pas imparables. Par la suite les contributions basées sur une analyse comportementale des entrées-sorties du système de fichiers, y compris celle que je propose section 5, reprennent les idées décrites ci-dessus. À noter que pour moi, les contributions de Young et Yung [112] et Kharraz *et al.* [89] sont les plus significatives et ce même en tenant compte des dernières publication de 2018.

CryptoLock (and Drop It) : Stopping Ransomware Attacks on User Data [103]

En 2016, Scaife *et al.* [103] présentent la première implémentation d'une contre-mesure qui surveille en temps réel les accès au système de fichiers. Les auteurs l'appellent CRYPTODROP.

4. Reproduit avec la permission de *Springer International Publishing*

Celle-ci s'intéresse uniquement aux modifications des données de utilisateur. Elle n'a pas pour vocation de remplacer les antivirus traditionnels mais de les complémenter là où ceux-ci échouent. De plus, cette solution n'assure pas l'intégrité de l'ensemble des fichiers de l'utilisateur, la vitesse de détection étant variable.

Trois indicateurs primaires et deux secondaires sont utilisés pour révéler un comportement malveillant. Lorsque chacun des trois indicateurs primaires s'est déclenché alors il est probable qu'une activité suspecte soit en cours. Les deux indicateurs secondaires agissent comme une sonde globale du système en cas de défaillance de la première ligne de défense. L'utilisation conjointe des trois indicateurs primaires permet de limiter les faux positifs. Les indicateurs primaires sont : (1) changement de type de fichier, (2) mesure de similarité et (3) entropie de Shannon.

Le premier indicateur s'appuie sur la librairie *libmagic* pour détecter le changement de type d'un fichier avant et après modification, c'est-à-dire lors d'une requête en écriture. Les auteurs observent que la mise à jour d'un logiciel peut entraîner le changement de type d'un fichier (i.e., nouveau standard).

Le second indicateur utilise une fonction de hachage d'un type particulier, appelé "similarity-preserving hash function". Le haché produit a la particularité de donner une information quant au contenu du fichier source, ainsi deux fichiers sources identiques auront des hachés semblables. On compare ensuite les hachés produits pour calculer un score. Ce score donne avec une certaine confiance une information sur le degré de similarité entre ces deux fichiers. Ici l'objectif est de déceler le chiffrement d'un fichier en interceptant la requête d'écriture malicieuse. En effet, le fichier chiffré est complètement différent du fichier original. Il est peu probable qu'une application légitime modifie complètement un fichier avec des données provenant d'une distribution très différente. La fonction de hachage utilisée ici est *sdhash* [5]. Une des limitations de *sdhash* est que la taille des fichiers doit être au minimum de 512 octets. De plus, l'interprétation des scores peut changer en fonction du type de fichier. Un score de 100 indique une très grande similarité entre deux fichiers. À l'inverse 0 signifie que la corrélation entre deux fichiers est identique à celle entre deux fichiers contenant chacun des données aléatoires. Les auteurs s'attendent donc à avoir des scores faibles lors des opérations de chiffrement sur des fichiers en clair.

Le dernier indicateur primaire est l'entropie de Shannon qui permet de déterminer le désordre associé à des données. Lors d'opérations de compressions ou de chiffrements, celle-ci est élevée car la probabilité associée à chaque symbole présent dans les données est proche de la distribution uniforme. La quantité d'information est donc maximale. Ici l'entropie est comprise entre 0 et 8 (i.e., $log_2(256)$). On la calcule sur une suite d'octets, soit 256 valeurs possibles pour un octet. Les deux indicateurs secondaires sont : (1) suppression massive et (2) "file type tunneling". La suppression massive de fichiers est considérée comme suspecte. Le terme "file type tunneling" désigne les applications qui ouvrent en lecture des fichiers de types différents pour finalement n'écrire qu'un seul et même type de fichiers. Ce comportement est typique des logiciels de rançon. Les auteurs proposent de faire la différence entre le nombre de types de fichiers ouverts en lecture et en écriture.

La détection est basée sur un premier score de réputation associé à chaque processus de

5. https://github.com/sdhash/sdhash

l'espace utilisateur qui fait l'union des trois indicateurs primaires. L'union des trois indicateurs permet de prendre une décision rapidement tout en limitant les faux positifs. En effet, les auteurs choisissent une valeur de seuil basse afin d'être réactif. De plus, il est peu probable d'observer une anomalie qui touche chacun des trois indicateurs alors que ceux-ci ont été conçus dans un but précis. On observe ainsi le comportement d'un processus au cours du temps. Un second type de score est utilisé, pour chaque indicateur, qu'il soit primaire ou secondaire, un score individuel lui est associé pour chaque processus. Les auteurs espèrent ainsi couvrir des logiciels de rançon aux comportements moins génériques. Il est possible pour un logiciel de rançon de rajouter des bits de remplissage à la fin de chaque fichier chiffré pour faire baisser l'entropie. Il s'agit là d'une course à l'armement et aux techniques d'évasions. Les auteurs avec ces 5 indicateurs obligent les auteurs de logiciels de rançon à mettre en place des moyens plus conséquents. Les scores sont incrémentés à chaque fois qu'un comportement suspect est détecté par l'un des indicateurs. Si un score dépasse la valeur de seuil alors on suspend les accès disques du processus incriminé dans l'attente d'une décision de l'utilisateur. Celui-ci est averti par une fenêtre. Les valeurs de seuil sont fixées de manière arbitraire et ne changent pas au cours du temps. De plus, la valeur de seuil utilisé pour l'union des trois indicateurs primaires n'est pas donnée dans le papier. Au vu des résultats ci-dessous, il me semble qu'il suffit que chacun des trois indicateurs soit suspect une seule fois pour lancer une alerte. La valeur de seuil pour le score de réputation propre à chaque indicateur (i.e., primaire et secondaire) est donnée, c'est 200.

Concernant son implémentation, CryptoDrop, ne protège que les documents placés dans le répertoire de l'utilisateur. Pour cela, il examine les requêtes en lecture et écriture. Très peu d'indications sont données quant à l'implémentation du pilote. En fait, les auteurs n'ont pas développé eux-mêmes le pilote. Ils ont utilisé une solution provenant d'une entreprise : EldoS. Le pilote fournit les informations requises en provenance du système de fichiers à une application dans l'espace utilisateur. Cette application, CryptoDrop, contient les mécanismes d'analyse et de détection. Ce pilote n'entraîne pas d'interférence avec des pilotes de chiffrement de disque comme la solution BitLocker. En effet, dans la pile des différents pilotes qui peuvent se greffer sur le système de fichiers, BitLocker est situé à une altitude inférieure et donc ne chiffre les données qu'après leur interception par le module de EldoS.

Avant de discuter des résultats, les auteurs font une observation : les notes de rançon possèdent une entropie faible. En effet, il s'agit d'un texte en anglais la plupart du temps intégré dans un format de fichier bien structuré. Ces notes de rançon sont présentent dans chaque dossier chiffré. Ce qui a pour conséquence de trop influencer l'indicateur de l'entropie de Shannon. Pour éviter que cela ne biaise la détection, ils associent un poids à chaque requête d'écriture en fonction de sa taille. Ainsi les notes de rançon qui sont de petites tailles (i.e., quelques Ko) n'auront qu'une influence mineure. Les explications sur la mise en place de cet indicateur ne sont pas claires. Il semble que les auteurs calculent une moyenne pondérée des requêtes en lecture (i.e., avant un éventuel chiffrement) et en écriture par processus en utilisant l'entropie de Shannon. Pour chaque processus ils évaluent ensuite la différence entre la moyenne de l'entropie de Shannon en lecture et écriture. Si la différence est supérieure à 0.1 alors l'indicateur est suspect.

Les expérimentations se déroulent avec la *Cuckoo Sandbox*, l'accès internet est restreint et l'image du système utilisé est un Windows 7. Chaque analyse dure 20 minutes, une fonction

de hachage cryptographique est utilisée pour vérifier l'intégrité du corpus utilisateur. Un effort particulier est apporté pour simuler un corpus utilisateur le plus vraisemblable possible. Pour cela, les auteurs utilisent des fichiers provenant de *DigitalCorpora*[6]. Ce site web fournit pour la recherche des contenus numériques (e.g., fichiers, images disques). Les auteurs disposent de 492 logiciels de rançon actifs appartenant à 14 familles.

Les résultats sont plus qu'excellent : CryptoDrop détecte ainsi 100% des logiciels de rançon. Pour la moitié des logiciels de rançon l'utilisateur perd au plus 10 fichiers. Un petit délai est notable si le logiciel de rançon commence par attaquer des fichiers clairs avec une grande entropie. En effet, la différence entre les requêtes de lecture et d'écriture n'est pas capturée par l'indicateur primaire (3). Il faut donc attendre qu'il attaque suffisamment de fichiers avec une entropie plus faible, ce qui dépend de l'organisation des dossiers de l'utilisateur. Au maximum c'est 30 fichiers perdus pour l'ensemble des échantillons. Le nombre de fichiers perdus est donc très faible, le corpus compte plus de 5000 fichiers. Pour 93% des échantillons l'union des trois indicateurs primaire est déclenché. Les auteurs indiquent que 63 échantillons déplacent le fichier original dans un répertoire temporaire pour le lire afin de générer sa version chiffrée. Cette technique peut suffire pour évader CryptoDrop. En effet deux descripteurs de fichier différents sont utilisés pour respectivement lire puis écrire. Parmi ces 63 échantillons 41 écrasent la version originale avec sa version chiffrée, ce qui permet à CryptoDrop de finalement faire le lien. Les 22 restants sont détectés grâce à l'indicateur secondaire qui surveille la suppression massive de fichiers. Les auteurs s'intéressent également à l'ordre dans lequel les dossiers sont explorés. La plupart des logiciels de rançon partent de la racine du disque puis font un parcours en profondeur. *CTB-Locker* à un comportement particulier. Une fois l'exploration terminée, il attaque les fichiers par ordre décroissant en fonction de leur taille. Dans ce cas, l'indicateur primaire (2) peut ne pas être disponible. En effet, il ne fonctionne pas si la taille des données est inférieure à 512 octets. *GPcode* quant à lui fait un parcours en largeur. L'ordre dans lequel les fichiers sont chiffrés (e.g., taille, entropie) peut faire varier la vitesse de détection de CryptoDrop.

Les auteurs font également des expérimentations avec des applications légitimes pour s'assurer que CryptoDrop ne déclenche pas de trop nombreux faux positifs. 30 applications sont ainsi testées. Seules deux applications sont identifées comme malicieuse : PGP et 7-Zip. Effectivement, celles-ci ont un comportement similaires à un logiciel de rançon. Elles lisent un ou plusieurs fichiers en clair puis écrivent un fichier avec une grande entropie au contenu et à l'extension différentes. Durant ces expérimentations aucune application hormis les deux ci-dessus n'a déclenché le score de réputation basé sur l'union des trois indicateurs primaires. Celui-ci est donc bien choisi pour les auteurs. La valeur de seuil propre à chaque indicateur est obtenue de manière empirique en observant la répartition des scores de réputation à la fin des tests. Ces applications sont testées une dizaine de minutes. Or, les scores de réputation augmentent avec le temps en fonction de l'activité de chaque processus. Ils vont donc tous lentement converger vers la valeur de seuil. Les auteurs reconnaissent qu'il est nécessaire de paramétrer une fenêtre glissante pour adresser ce problème. Ils laissent cela comme un travail futur. Tout en précisant que celle-ci pourrait être abusée par un logiciel de rançon (e.g., chiffre lentement).

6. https ://digitalcorpora.org/corpora/files

Les limitations de CryptoDrop sont nombreuses. Du fait de l'utilisation d'un pilote tiers pour surveiller les accès au système de fichiers, la solution est inutilisable. En effet, CryptoDrop de l'aveu même des auteurs paralyse le système. La conception est à revoir. La communication entre l'espace utilisateur et le noyau est trop coûteuse. Les auteurs indiquent une latence de $9ms$, $16ms$ et $1ms$ pour respectivement les opérations d'écriture, de renommage et de lecture. Aucune explication n'est donnée pour expliquer le contexte dans lequel cette latence a été mesuré nie comment. De plus, le binaire de la solution n'est pas fournit. Il n'est donc pas possible d'utiliser CryptoDrop pour le comparer et reproduire les expérimentations. Le fait de placer le module d'analyse et de prise de décision dans l'espace utilisateur rend la solution vulnérable. Elle pourrait être désactivée par un logiciel malveillant. Pour Kharraz *et al.* [89] la contre-mesure se devait de résider dans le noyau, or ici ce n'est pas le cas. Il ne s'agit que d'une implémentation partielle de la solution de Kharraz *et al.* [89].

Néanmoins, CryptoDrop est la première contre-mesure à surveiller les entrées-sorties du système de fichiers, son implémentation donne de très bon résultat. Le taux de détection est de 100%. Le nombre de fichiers perdus est minime et les faux positifs eux aussi très peu nombreux. En fait, cette solution est un bon exemple qui montre bien que pour être viable une solution doit faire des compromis notamment pour assurer un faible impact sur le système. En effet, CryptoDrop surveille pas moins de 5 indicateurs. N'est-il pas possible de faire aussi bien avec autant voir moins d'indicateurs tout en ayant un faible impact ? Les contributions suivantes vont tenter de répondre à cette question.

UNVEIL : A Large-Scale, Automated Approach to Detecting Ransomware [87]

En 2016, Kharraz *et al.* [87] présentent une seconde contribution après [89]. Il s'agit cette fois d'une plateforme d'analyse dynamique de logiciels de rançon appelé UNVEIL. Ce n'est pas une contre-mesure temps réel mais les moyens mis en œuvre pour analyser et détecter les logiciels de rançon. Les auteurs indiquent que les solutions d'analyse existantes ne sont pas adaptées à la détection et à la classification des logiciels de rançon. En effet, ceux-ci ont un comportement qui peut paraître similaire à une application bénigne (e.g., compression). De plus, pour distinguer des échantillons de familles différentes, il est nécessaire de s'intéresser à des comportements spécifiques qui ne sont pas pris en compte par les plateformes actuelles, d'où les problèmes de classification. Kharraz *et al.* [87] ont mis au point une plateforme d'analyse qui va automatiquement créer un environnement d'exécution réaliste puis enregistrer les interactions du logiciel malveillant sur cet environnement. La surveillance se fait avec la granularité d'un processus en s'intéressant comme Kharraz *et al.* [89] l'avaient eux-même proposés en 2015 au comportement du système de fichiers. La détection se fait post-mortem à partir de la trace générée pendant la phase d'analyse. Les auteurs présentent également un moyen de détecter post-infection les logiciels de rançon qui se contentent de bloquer l'écran (i.e., screen locker), c'est-à-dire qui ne touchent pas aux fichiers de l'utilisateur. Je ne m'y intéresse pas, car ce comportement ne modifie pas les données et peut donc être plus facilement déjoué. L'architecture de UNVEIL est visible figure 2.3, notamment la surveillance des I/Os sur le système de fichiers.

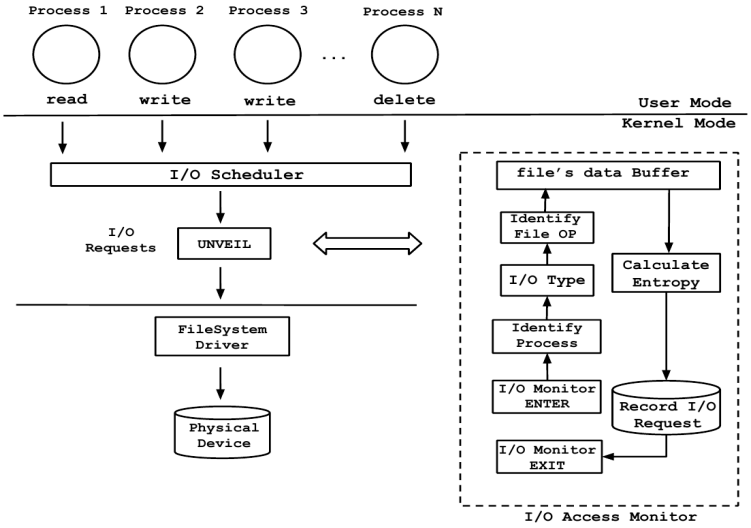

FIGURE 2.3 – (Kharraz *et al.* [87] [7]) Architecture de UNVEIL.

Les auteurs jugent que les logiciels de rançon peuvent utiliser les mêmes stratégies que les logiciels malveillants "classiques" pour : évader la détection, se propager et attaquer un utilisateur. UNVEIL considère une attaque comme réussie si une des trois activités suivantes s'est déclarée : (1) message persistant sur le bureau, (2) chiffrement ou suppression des fichiers utilisateurs et (3) même activité que (2) mais en considèrant que l'opération s'est déroulée de manière sélective en s'appuyant sur des attributs (e.g., taille, date).

UNVEIL est centré autour de deux axes. Le premier est la génération d'un environnement d'exécution réaliste, unique et non déterministe pour les logiciels de rançon. En effet, il est essentiel d'empêcher ceux-ci de détecter qu'ils se trouvent dans un environnement d'analyse (i.e., fingerprinting). Des caractéristiques propres à chaque système d'analyse peuvent être répertoriées et rendues publiques. Le défi est double. Il faut donc être capable de générer automatiquement un environnement qui ne soit pas reconnaissable et qui paraisse réaliste. L'objectif est d'inciter le code malicieux à s'exécuter.

Le second axe de UNVEIL est la surveillance des entrées-sorties sur le système de fichiers. Pour cela, les auteurs implémentent un pilote de type "file system minifilter driver". Pour chaque type de requête, UNVEIL dispose d'une fonction de contrôle qui lui est associée afin de générer des traces les plus exhaustives possible. Les auteurs ont donc une visibilité complète sur les I/Os provenant de l'espace utilisateur. Les interactions des processus utilisateurs avec le système de fichiers sont formalisées en séquence d'accès. Ces séquences d'accès sont représentées par les I/Os

7. Reproduit avec la permission de *USENIX Association*

capturées dans les traces. Une trace T est une séquence de t_i :

$$\begin{aligned} & t_i = \langle P,\ F,\ O,\ E \rangle \\ & P \textit{ est l'ensemble des processus utilisateurs} \\ & F \textit{ est l'ensemble des fichiers touchés} \\ & O \textit{ est l'ensemble des } I/Os \\ & E \textit{ est l'entropie associée à chaque requête de lecture et d'écriture} \end{aligned} \tag{2.3}$$

Les stratégies misent en place par les logiciels de rançon sont donc bien visibles. Pour chaque fichier chiffré, la même stratégie est rejouée X fois. UNVEIL calcule l'entropie de Shannon pour chaque requête de lecture et écriture. Les auteurs comparent ensuite l'entropie des requêtes de lecture et d'écriture sur les mêmes fichiers aux mêmes positions pour détecter un comportement anormal. Une trace est obtenue pour chaque échantillon. Cette trace est ensuite triée en fonction du nom des fichiers touchés et de l'horodatage des I/Os. Une fois les I/Os triées en fonction des fichiers touchés, les auteurs peuvent observer les répétitions engendrées par le ou les processus malicieux. La présence dans les séquences d'accès des fichiers de requête d'écriture et de suppression est un critère spécial utilisé pour la détection. Les auteurs ont identifié dans une phase d'analyse préliminaire trois séquences d'accès malicieuses : (A) chiffrement en place, (B) création de la version chiffrée en parcourant en lecture le clair puis suppression du clair et (C) même chose que (B) mais en écrasant le clair à la fin. La FIGURE 2.4 présente ces trois motifs caractéristiques.

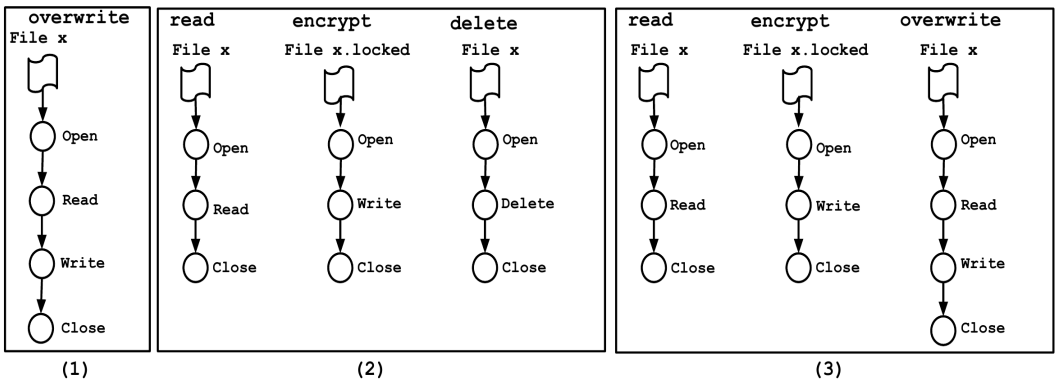

FIGURE 2.4 – (Kharraz *et al.* [87] [8]) Les trois stratégies les plus répandues parmi les logiciels de rançon pour attaquer les fichiers utilisateur.

L'environnement de test est généré automatiquement en considérant 4 ensembles d'extensions ciblés par les logiciels de rançon : documents, clés, licences et archives. Tout d'abord les auteurs créent un nombre égal de fichiers pour chaque extension. Le nombre de ces fichiers est choisi aléatoirement. Puis pour ajouter de l'entropie, un nombre aléatoire de fichiers différents est ajouté pour chaque extension. L'environnement dispose également de propriétés supplémentaires : nom des fichiers et des répertoires tirés d'une liste de mots, modification des attributs des fichiers,

8. Reproduit avec la permission de *USENIX Association*

création des répertoires à des profondeurs différentes, création des fichiers à la volée, etc. Le fait que UNVEIL soit non déterministe le rend difficile à reconnaître pour un logiciel malveillant. L'implémentation du pilote du type "file system minifilter driver" est quant à lui composé de 2800 lignes de code source en C++. Celui-ci intercepte toutes les I/Os provenant de l'espace utilisateur du type I/O request packet. Il s'agit d'une structure de données qui peut encapsuler les requêtes d'entrées-sorties du système de fichiers. C'est la plus utilisée par les applications utilisateurs.

Les expérimentations ont pour objectif de démontrer que UNVEIL est capable de détecter des logiciels de rançon connus mais aussi inconnus jusqu'alors. La plateforme d'analyse *Cuckoo Sandbox* est utilisée comme base pour UNVEIL. En effet, celle-ci fournit des services bien pratiques (e.g., gestion de machines virtuelles) pour lancer des analyses à grande échelle. Les auteurs rapportent que n'importe quelle autre plateforme d'analyse (e.g., *BitBlaze* [3]) aurait pu être employée. Au total, c'est 56 machines virtuelles équipées de Windows XP et de plusieurs disques NTFS qui ont été utilisés pour les analyses. Les communications avec l'Internet sont autorisées, principalement pour permettre aux échantillons de joindre leur serveur de commande et de contrôle. Chaque analyse dure 20 minutes, pendant lesquelles les interactions sur le système de fichiers sont enregistrées et constituent une trace. Des interactions avec l'interface utilisateur sont également générées aléatoirement (e.g., navigation, clic) pour simuler une utilisation normale. Bien entendu, chaque analyse contient des fichiers utilisateurs créés à partir des informations décrites dans le paragraphe précédent. Les auteurs disposent d'un corpus de 319 logiciels de rançon (i.e., qui chiffrent les données) répartis en 8 familles. Ce corpus est qualifié de "ground truth". En effet, il est constitué d'un ensemble de logiciels de rançon représentatif du monde réel. Les séquences d'accès malicieuses sur les fichiers identifiés plus tôt doivent être capables de les détecter. Les logiciels de rançon sont correctement détectés lors de cette phase de vérification. Les auteurs rapportent un comportement particulier des échantillons *CryptoWall 4.0*. Ceux-ci modifient le nom et l'extension du fichier original en les remplaçant par des caractères aléatoires. Une telle technique diminue les chances de retrouver les fichiers compromis grâce à leur nom dans la MFT du système de fichiers NTFS.

Après avoir testé des échantillons marqués comme malicieux, les auteurs font une nouvelle expérimentation pour évaluer le taux de faux positifs. Un nouveau corpus constitué de 149 applications bénignes et 384 programmes malveillants (i.e. pas de logiciels de rançon) est crée. Des applications bénignes imitant le comportant d'un logiciel de rançon sont sélectionnées (e.g., SDelete, Winzip, DiskCryptor). Les auteurs discutent de la stratégie à adopter lorsque des requêtes d'accès suspectes sont capturées par UNVEIL. Il est possible de distinguer très simplement une application bénigne d'un logiciel de rançon pour les auteurs. Dans le premier cas, le fichier original n'est jamais altéré (i.e., modifié ou supprimé). Tout du moins, c'est l'hypothèse raisonnable que les auteurs font concernant le comportement par défaut des applications. Ainsi si une application légitime est suspecte et qu'en plus celle-ci supprime de manière délibérée le fichier original alors elle est marquée comme malicieuse. En pratique, l'utilisateur n'a pas intérêt à supprimer le fichier original, ce serait prendre des risques. À partir des traces obtenues lors des expérimentations et de l'hypothèse ci-dessus, les auteurs concluent que les programmes bénins produisent par défaut des I/Os bien différentes de celles générées par des logiciels de rançon. De plus, il est clair qu'un logiciel de rançon entraîne de nombreuses répétitions d'une séquence d'accès malicieuse

pour accèder aux fichiers utilisateurs. Une application légitime, quant à elle, ne génére une telle séquence que pour un nombre faible de fichiers, voir même un seul.

Après avoir vérifié les capacités de détection et évaluer les faux positifs de leur contre-mesure, les auteurs procèdent à une troisième expérimentation. Celle-ci à pour objectif de vérifier la justesse de prédiction de UNVEIL sur des programmes malveillants récents non marqués. Le corpus est constitué de plus de 148k échantillons. La vérification des résultats est une tâche ardue pour cette expérimentation, en effet les échantillons ne sont pas marqués et très nombreux. Les auteurs procédent à une vérification manuelle des traces contenant un processus marqué comme malicieux. Pour rappel, un processus est malicieux s'il exhibe une des trois séquences d'accès identifiées comme telle précédemment. L'entropie de Shannon n'est utilisée que pour aider "l'opérateur" à prendre une décision quant à la nature de l'échantillon. En effet, la création de nombreux fichiers avec une entropie élévée est un indice d'une attaque pour les auteurs. Même si le logiciel de rançon procède à une suppression sécurisée du contenu original (i.e., entropie faible) avant de le réécrire, il doit préalablement créer la version chiffrée. Les auteurs ne trouvent pas de faux positifs en analysant manuellement les résultats de UNVEIL. Une évaluation exacte des faux négatifs n'est pas possible, en effet il y a trop d'échantillons. Les auteurs font donc une approximation. Les échantillons non détectés comme des logiciels malveillants par *VirusTotal* ne sont pas considérés comme malicieux. Cela permet d'éliminer 47% des échantillons. Pour les échantillons restant les auteurs analysent manuellement les traces à la recherche d'indices comme précédemment pour les faux positifs. Les auteurs ne trouvent pas de faux négatifs. Sur plus de 148k échantillons, 9.2% sont détectés comme des logiciels de rançon (i.e., "locker" et "cryptor"). Parmi ces 13637 logiciels de rançon, 7572 soit 55.5% chiffrent les données de l'utilisateur. UNVEIL possède donc un taux de détection de 100% tout en assurant un taux de faux positifs de 0% à la suite des expérimentations sur des échantillons marqués et non marqués.

Un des objectifs de UNVEIL est de détecter automatiquement des échantillons inconnus jusqu'alors (i.e., zero-day). Les auteurs collectent ainsi quotidiennement des échantillons qu'ils soumettent à UNVEIL et *VirusTotal*. Si un échantillon est détecté comme un logiciel de rançon par UNVEIL alors les auteurs regardent si celui-ci a également été détecté par *VirusTotal*. Si l'échantillon n'est pas détecté par *VirusTotal* alors il s'agit d'une nouvelle détection. La soumission des échantillons ici est concurrente de l'expérimentation précédente. C'est donc 148k échantillons soumis pour 13637 logiciels de rançon détectés par UNVEIL. En plus de s'intéresser à de nouvelles détections, les auteurs veulent observer le temps supplémentaire qu'il faut aux antivirus pour découvrir les échantillons nouvellement détectés par UNVEIL. Pour cela, ils soumettent quotidiennement pendant 6 jours à *VirusTotal* les logiciels de rançon nouvellement détectés. Les résultats sont éloquents. Sur les 13637 logiciels de rançon détectés par UNVEIL, 9872 soit 72.2% sont de nouvelles détections. De plus, pour leur première soumission à *VirusTotal* ces 9872 logiciels de rançon ne sont détectés par aucun des 55 scanners d'antivirus. Après 6 jours, les résultats des antivirus sont concentrés de deux manières : (1) taux de détection de $\sim 100\%$ et (2) taux de détection faible. UNVEIL est donc capable d'identifier de nouveaux logiciels de rançon, c'est-à-dire 9872 pour l'expérimentation ci-dessus.

Les auteurs discutent également des limitations. La plus importante est le fait que la preuve de concept présentée ici soit basée sur la *Cuckoo Sandbox*. L'utilisation d'un environnement

virtualisé introduit de nombreux artefacts qui peuvent être détectés. Les auteurs ajoutent néanmoins que UNVEIL est agnostique quant à la plateforme d'analyse dynamique choisie. Il est toujours possible pour un programme malveillant de détecter que l'environnement d'exécution créé automatiquement est en fait factice. Par contre, cela obligent les logiciels malveillants à déployer davantage d'efforts et donc à retarder l'attaque, ce qui n'est pas souhaitable. Concernant les accès au système de fichiers, les logiciels de rançon peuvent se contenter de ne chiffrer qu'une partie bien spécifique pour rendre le fichier illisible. Même dans ce cas, il est toujours nécessaire d'ouvrir en écriture le fichier pour le modifier. Ce comportement est visible par UNVEIL. Il faudrait alors ajouter la séquence d'accès de ces I/Os aux 3 séquences déjà répertoriées comme malicieuses. Par contre cela pourrait induire des faux positifs, mais le risque reste faible car pendant l'analyse d'un échantillon l'activité du système est contrôlée. Les auteurs discutent également de la possibilité pour un logiciel de rançon d'annihiler le pilote en charge de la surveillance des I/Os. Un logiciel de rançon doit alors soit : (1) disposer des priviléges administrateurs ou (2) exploiter une faille du noyau. Les deux cas de figure sont peu probables. De plus, les logiciels de rançon n'ont nul besoin de s'exécuter dans le noyau, l'espace utilisateur leur permet déjà d'atteindre leur but. Deux métriques importantes ne sont pas données par les auteurs : (1) la taille moyenne d'une trace en fonction du temps et de l'activité et (2) le temps nécessaire pour créer un environnement automatiquement.

Pour conclure, UNVEIL est la première plateforme d'analyse dynamique développée spécifiquement pour les logiciels de rançon. Ces propriétés lui permettent d'inciter les logiciels de rançon à s'exécuter. Les auteurs découvrent ainsi pas moins de 9872 échantillons non détectés par les antivirus de *VirusTotal*. De plus, au cours des expérimentations trois séquences d'accès aux fichiers bien spécifiques des logiciels de rançon sont isolées. Le recours à certains motifs tels que des requêtes d'écriture et de suppression et la répétition de ces motifs sont des indices d'une attaque. Les trois séquences d'accès malicieuses permettent à UNVEIL d'obtenir un taux de détection de 100% et de faux positifs de 0% en analysant les traces post-mortem. Une des clés pour distinguer un programme légitime avec un comportement suspect (i.e., compression) d'un logiciel de rançon est le fait que le premier normalement n'altére jamais le fichier original. Cette solution pour détecter les logiciels de rançon post-mortem n'est pas transposable à une solution temps réel. En effet, la détection est basée sur la répétititon de séquences suspectes. En temps réel, trop de fichiers seraient perdus avant qu'une décision ne soit prise. De plus, le coût pour le système de surveiller l'ensemble des I/Os, c'est-à-dire des IRPs ici n'est pas réaliste.

Automated Dynamic Analysis of Ransomware : Benefits, Limitations and use for Detection [105]

En 2016, Sgandurra *et al.* [105] présentent une contre-mesure, EldeRan, qui adresse le problème de la détection des logiciels de rançon en utilisant de l'apprentissage automatique supervisé. Les objectifs de cette contre-mesure sont duals : (1) détection automatique post-mortem via des traces d'exécutions et (2) détection temps réel sur la machine d'un utilisateur. L'approche pour identifier un comportement malveillant est ici un peu différente. Les auteurs ne s'intéressent au comportement d'un processus pendant les 30 secondes qui suivent son lancement.

Le but est d'entraîner un modèle capable de détecter un logiciel de rançon le plus tôt possible et ainsi de ne perdre que peu de fichiers. Les auteurs évaluent également la capacité de EldeRan à détecter de nouveaux échantillons. EldeRan est ainsi comparé avec *VirusTotal*.

Une des parties critiques lors de l'utilisation de l'apprentissage automatique est la sélection des caractéristiques. Les auteurs se concentrent uniquement sur des actions qui se déroulent lors du lancement d'un processus (i.e., 30 secondes). Pour cela, la première partie du travail consiste à récolter des traces d'exécution de logiciels de rançon et de programmes légitimes à partir de la *Cuckoo Sandbox*. Les caractéristiques ou ensemble de caractéristiques suivantes sont collectées pour chaque analyse :

— Appels Windows API,

— Opérations sur les clés de registres,

— I/Os sur le système de fichiers,

— Opérations réalisées pour chaque type de fichiers,

— Opérations sur les répertoires,

— Liste des fichiers "déposés" (i.e., dropped),

— Chaînes de caractères embarquées dans un binaire.

Au total, les auteurs ont analysé 582 logiciels de rançon et 942 applications légitimes. Parmi les logiciels bénins, les auteurs sélectionnent une large variété d'applications. Les expérimentations se déroulent avec une image de Windows XP, la présence de fichiers utilisateurs et une connexion Internet. Durant les analyses une activité minimale est simulée. Une fois les données récupérées, les auteurs procèdent à la sélection des caractéristiques les plus significatives. Un pré-traitement est réalisé sur les traces qui sont au format JavaScript Object Notation (JSON). En effet, un ensemble de matrices est créé à partir des caractéristiques énoncées ci-dessus. Par exemple, une matrice des appels à l'API Windows est créée, celle-ci dénote la présence ou l'absence d'une fonction lors d'une analyse. Un comportement analogue est réalisé pour les autres caractéristiques. Ce qui permet finalement d'ajouter de nombreuses combinaisons. Au total, après avoir analysé les logiciels bénins et malveillants, les auteurs recensent 30967 caractéristiques. La sélection est réalisée en utilisant l'information mutuelle. Plus cette quantité d'information est importante pour une caractèrisque donnée et plus celle-ci est discriminante. Dans le cas présent, il s'agit de distinguer la classe des applications bénines et malveillantes. Les auteurs ne donnent pas les caractéristiques les plus significatives, ce qui est dommage. Ils regroupent les caractéristiques en fonction de leur ensemble d'appartenance (e.g., Appels Windows API) en considérant les 400 et 100 plus significatives. Ils donnent ensuite le pourcentage de ces caractéristiques appartenant à chaque ensemble.

Il est surprenant de voir tableau 2.1 que les opérations sur le système de fichiers ne contribuent que pour 5.25% et 6% des caractéristiques les plus significatives. Ce résultat est somme toute logique, car les auteurs ne s'intéressent qu'aux 30 premières secondes de la vie des processus. Les logiciels de rançon n'ont tout simplement pas commencé à chiffrer.

La phase de sélection des caractéristiques est maintenant terminée. Les auteurs testent les capacités de classification de EldeRan, c'est-à-dire "ransomware" ou "goodware". Un modèle par

Tableau 2.1 – (Sgandurra *et al.* [105] [9]) Contribution de chaque "ensemble" pour le top 400 et 100 des caractéristiques les plus discriminantes.

Ensembles	Top 400	Top 100
Opérations sur les clés de registres	48.25%	49%
Appels Windows API	24%	27%
Chaînes de caractères	8.5%	5%
Opérations réalisées pour chaque type de fichiers	8%	9%
I/Os sur le système de fichiers	5.25%	6%
Opérations sur les répertoires	4%	2%
Liste des fichiers "déposés"	2.25%	2%

régression logistique est employé. Ces modèles permettent de modéliser des variables binaires, par exemple absence ou présence d'une clé de registre. Un tel modèle permet de relier un événement (i.e., logiciel de rançon) à une combinaison de variables (i.e., caractéristiques). La régression logistique est un cas particulier de la régression linéaire. Ces modèles sont simples à entraîner et réentraîner dynamiquement. De plus, même lorsque le nombre de caractéristiques N est important, les performances restent acceptables. Les auteurs comparent les performance de EldeRan avec un modèle bayésien et un modèle de machines à vecteurs de support. Ils complètent cette comparaison avec les 5 antivirus de *VirusTotal* les plus performants sur les échantillons de tests.

Cette expérimentation fait également varier le nombre de caractéristiques N (i.e., entre 50 et 1500). Le jeu de données obtenu précédemment est découpé en 100 groupes aléatoires pour chaque valeur de N dont 80% est dédié à l'entraînement et 20% au test. Les modèles sont évalués en traçant la moyenne de leur fonction d'efficacité du récepteur (i.e., courbe ROC). Une courbe ROC donne le taux de vrais positifs en fonction du taux de faux positifs. Les résultats montrent que le modèle bayésien est de loin le moins performant. Sans doute parce que celui-ci considère que les caractéristiques sont indépendantes. EldeRan et le modèle de machines à vecteurs de support ont des performances très proches, mais EldeRan est légérement meilleur. Ces deux modèles prennent en compte les éventuelles dépendances entre les caractéristiques. Un autre indice est utilisé pour évaluer les performances, il s'agit de l'aire sous la courbe ROC (i.e., AUC). Celui-ci permet de mesurer la qualité des prédictions. Les meilleurs résultats sont obtenus avec 400 caractéristiques. Au delà, cela n'améliore en rien les prédictions. EldeRan obtient un taux de détection de 96.3% et de faux positifs de 1.61%. Le taux de détection est en moyenne de 95.4% pour les 5 antivirus de *VirusTotal* les plus performants. Ceux-ci ont un taux de détection plus faible, mais leurs taux de faux positifs est de 0% pour quatre d'entre eux. Les auteurs ajoutent que les échantillons faisant partie des ensembles de tests n'ont jamais été analysés précédemment par leur algorithme. Les antivirus eux fonctionnent différemment, ils possèdent des signatures pour les échantillons déjà étudiés. Or, les échantillons du corpus sont tous probablement connus des antivirus. Les auteurs pensent que la comparaison favorise *VirusTotal*. EldeRan obtient un

9. Reproduit avec la permission des auteurs

taux de faux positifs qui n'est pas négligeable selon moi car pendant les analyses une activité minimale est simulée. De plus, les faux positifs ne sont pas investigués.

EldeRan n'est pas exempt de certaines limitations. En effet, un logiciel de rançon peut attendre un temps non déterminé avant d'exhiber son comportement. Il peut aussi détecter qu'il s'exécute dans un environnement contrôlé, et ainsi empêcher l'extraction des caractéristiques. De plus, l'évaluation des faux positifs est incomplète. Les auteurs n'ont pas analysé de logiciels malveillants pour vérifier que EldeRan est capable de les différencier des logiciels de rançon. Au vu des deux ensembles de caractéristiques les plus significatifs, c'est-à-dire les appels à l'API Windows (i.e., 27%) et les clés de registres (i.e., 49%) ont peu en douter. En effet, il s'agit de caractéristiques semblables entre les logiciels malveillants et de rançon. C'est sans doute pour cela que les auteurs ajoutent qu'il faudrait incorporer au modèle les séquences d'accès malicieuses des logiciels de rançon aux fichiers. EldeRan s'appuie sur des traces obtenues lors d'une analyse dynamique, la détection est donc post-mortem. Les auteurs ajoutent qu'une détection temps réel est possible mais n'évaluent pas le coût pour le système. Le fait que 400 caractéristiques soient nécessaire pour obtenir un taux de détection de 96.3% est sans aucun doute trop coûteux en temps réel.

L'approche proposée par EldeRan est originale, c'est-à-dire extraire des 30 premières secondes d'un processus des actions suspectes. Seulement celle-ci a ses limites. Il n'en reste pas moins que EldeRan est la première plateforme d'analyse dynamique à proposer une détection basée sur de l'apprentissage automatique supervisé pour détecter les logiciels de rançon.

An Efficient Approach to Detect TorrentLocker Ransomware in Computer Systems [96]

En 2016, Mbol *et al.* [96] présentent un nouvel indicateur de compromission. Il s'agit de la divergence de Kullback-Leibler. La contribution de Scaife *et al.* [103], CryptoDrop, utilise parmi ses indicateurs : l'entropie de Shannon et une fonction de hachage préservant la similarité. Cette dernière est jugée trop couteûse, les auteurs souhaitent donc la remplacer. Mbol *et al.* [96] ne présentent pas d'implémentation d'une contre-mesure et ne réalisent aucun test avec des logiciels de rançon. Il s'agit d'expérimentations "hors-ligne" pour tester les capacités de leur indicateur.

Les auteurs constatent que l'entropie de Shannon n'est pas capable de distinguer des transformations malicieuses sur certains formats de fichiers. En effet, les fichiers qui possèdent déjà une entropie éléve (e.g., archive) sont sources de faux positifs. Les auteurs s'intéressent au format de fichier JPEG. Le format JPEG est un bon exemple de fichier structuré à l'entropie élevé, car celui-ci est très employé. Ce format de fichier contient des blocs de données compressés. Le simple fait de manipuler des images avec un logiciel d'édition photo peut suffire à déclencher un faux positif. Il faut donc être capable de distinguer des fichiers structurés à l'entropie élevées des fichiers chiffrés (i.e., non structurés à l'entropie élevées). Si la la divergence de Kullback-Leibler est capable de réaliser cela, alors celle-ci peut faire de même pour des formats de fichiers à l'entropie plus faible.

Les auteurs calculent l'entropie de Shannon sur 150 images claires puis leurs versions chiffrées. Le résultat est qu'en moyenne l'entropie est 7.9684 et 7.9998 pour respectivement : les images

en clair et chiffrées. Il est donc très difficile de les discerner. Cette distinction a été réalisée par l'utilisation de la fonction de hachage préservant la similarité dans CryptoDrop. Une seconde expérience est réalisée avec 2000 images et leur chiffrées. Cette fois, les auteurs comparent les valeurs obtenues en utilisant la divergence de Kullback-Leibler. La divergence de Kullback-Leibler est la différence entre l'entropie croisée et l'entropie de Shannon. L'entropie croisée correspond au nombre de bits moyen nécessaire pour transmettre une information qui suit une loi de probabilité p si l'on utilise une loi de probabilité q (i.e., plus ou moins optimisée) pour coder l'information. La différence est une "distance" entre deux distributions. Les auteurs souhaitent ici déterminer la distance entre la distribution des données observées q (i.e., une image) et la distribution uniforme p (i.e., un chiffré). Les auteurs s'intéressent ici à la distribution des données avec la granularité d'un octet. Les résultats montrent qu'en moyenne la divergence de Kullback-Leibler est de 0.01 pour les images en clair et de 0.001 pour les images chiffrées. Les auteurs déterminent ensuite une valeur de seuil α pour séparer les deux flux de données. Pour une valeur $\alpha = 0.007$, la qualité de précision (i.e., vrai positif et négatif) est de 99.98%. Avec ce paramètrage, les auteurs ne déclencheraient qu'un seul faux positif, c'est-à-dire un fichier chiffré détecté comme une image. Le plus important dans l'hypothèse du déploiement dans une solution temps réel est qu'ici toutes les images sont correctement détectées sans qu'aucunes ne soient classées comme malicieuse. Les auteurs reproduisent la même expérience en se s'intéressant qu'aux premiers 128 Ko, 256 Ko et 512 Ko des fichiers. Les résultats montrent qu'en ne s'intéressant que de manière partielle aux images il est toujours possible de distinguer les images claires et chiffrées. L'avantage bien entendu est le gain de temps pour le système.

Les auteurs proposent donc de remplacer la fonction de hachage préservant la similarité utilisée par Scaife *et al.* [103] par la divergence de Kullback-Leibler. Celle-ci est capable d'identifier des transformations malicieuses, c'est-à-dire des opérations de chiffrement sur des fichiers clairs. Pour distinguer les opérations de chiffrement légitimes d'un utilisateur et malicieuses d'un logiciel de rançon il faut davantage d'indices. C'est pour cela que la divergence de Kullback-Leibler ne peut être utilisée seule. De plus, nous aurions aimé voir comment se comporte la divergence de Kullback-Leibler sur des fichiers compressés. Les auteurs indiquent que ceux-ci pourraient déclencher de fausses alertes.

ShieldFS : A Self-healing, Ransomware-aware Filesystem [71]

En 2016, Continella *et al.* [71] présentent indépendamment de Scaife *et al.* [103] une solution appelée ShieldFS. Celle-ci est également basée sur une surveillance temps réel des entrées-sorties sur le système de fichiers et étend la contribution de Scaife *et al.* [103]. En effet, ShieldFS implémente un mécanisme de restauration des fichiers compromis totalement transparent, intégré dans le pilote. De plus, un outil permettant de retrouver des clés secrètes dans la mémoire d'un processus est aussi présent. Cet outil est optionnel et ne fait pas parti des indicateurs de compromission. Une attention particulière est notée pour évaluer l'impact de ShieldFS.

Les auteurs souhaitent modéliser le comportement courant du système de fichiers dans différents cas d'usages (e.g., bureautique, développement). Pour cela, ils développent leur propre outil : IRPLogger. Celui-ci va enregistrer les opérations du type IRP en provenance de l'espace

utilisateur. IRPLogger est un pilote de type "file system minifilter driver". Les auteurs ajoutent des informations non présentes dans une IRP lors de la phase de collecte. Pour chaque IRP, les informations suivantes sont enregistrées :

$$< time,\ program\ name,\ PID,\ IRP\ op,\ entropy,\ file\ info > . \tag{2.4}$$

Cet outil est déployé sur les machines de 11 volontaires afin d'enregistrer leurs activités journalières. Au total, un ensemble de référence de 28 *Go* est obtenu. Ce qui correspond à pas moins de 1.7 milliard d'IRPs générées par plus de 2000 applications différentes. Le déploiement de IRPLogger s'étend de $8h$ pour la capture la plus courte sur une machine, à $215h$ soit presque 9 jours pour la plus longue. Des expérimentations similaires sont ensuite menées avec des logiciels de rançon. Les auteurs souhaitent ainsi comparer la distribution d'événements en particuliers (i.e., indicateurs de compromission). Un jeu de données de 19.7 *Go* est obtenu grâce à la collecte de 383 logiciels de rançon répartis en 5 familles. En tout c'est donc 663 millions d'IRPs collectées pour caractériser une activité malveillante. La période d'écoute dure 90 minutes pour chaque logiciel de rançon et à lieu dans une machine virtuelle équipée de Windows 7 64-bit connectée à Internet. Les auteurs distinguent les modifications apportées aux données de l'utilisateur et celles qui sont réalisées dans des répertoires du système. De nombreuses IRPs sont générées par des processus du système aux comportements particuliers (e.g., service mise à jour). Ils excluent par exemple les répertoires *WINDOWS* et *Program Files* des dossiers utilisateurs.

A partir de l'analyse des données collectées, les auteurs adressent le problème de la détection en utilisant un algorithme d'apprentissage automatique supervisé. L'entraînement de ce modèle est accompli à partir des données collectées et d'une liste de caractéristiques (i.e., features). Les auteurs identifient 6 indicateurs de compromission :

1. énumération des fichiers d'un répertoire,
2. nombre de fichiers lus,
3. nombre de fichiers écrits,
4. nombre de fichiers renommés ou déplacés,
5. nombre de fichiers manipulés différents,
6. l'entropie de Shannon associée à chaque requête en écriture.

Différents modèles d'apprentissage sont utilisés. En effet, un logiciel de rançon peut exhiber son comportement malveillant dans un processus mais également à travers différents processus. L'objectif est de passer à travers une contre-mesure n'ayant pas pris en compte cette possibilité tout en étant plus efficace. Un premier ensemble de modèles "process-centric" est entraîné sur les processus, chacun pris séparément. Aucune explication n'est donnée pour expliquer comment les labels des processus appartenant à une trace malicieuse ont été obtenu (e.g., manuellement). Le second ensemble est "system-centric", il considère les IRPs d'une trace comme venant d'un seul et unique processus. Nous appelons une trace, les données collectées provenant de l'analyse d'un logiciel de rançon ou d'une capture utilisateur. Les modèles orientés système ont pour objectif de détecter un logiciel de rançon multi-processus. Les auteurs indiquent néanmoins que ceux-ci

ne peuvent être utilisés seuls, ils déclencheraient trop de faux positifs. Ils sont donc employés en combinaison des modèles dédiés aux processus.

La classification d'un processus comme bénin ou malicieux se fait à des intervalles fixes. Ces intervalles, les "ticks", sont fonction du nombre de fichiers touchés par un processus. Les auteurs prédéterminent ces intervalles afin d'obtenir un compromis entre la rapidité de détection et le système de restauration qui consomme des ressources. Ainsi par exemple, quand un processus vient d'accéder à 1% des données utilisateurs, celui-ci a atteint le premier "tick". Ce qui déclenche une prise de décision des modèles "process-centric" entraînés sur 1% du corpus des IRPs. Les auteurs font de même pour les autres "ticks". Ils disposent ainsi d'un tableau contenant des modèles représentant chacun un pourcentage différent des fichiers accédés. Le jeu de données est découpé en 28 intervalles. La taille de ces intervalles augmentent de manière exponentielle en fonction du nombre de fichiers touchés par un processus. Pour prendre en compte les changements de comportement des processus, les auteurs surveillent également leurs activités à plus ou moins court et long terme. En effet, la quantité de données prisent en compte lors d'un "tick" par les modèles orientés processus est variable. Celle-ci peut varier de N "ticks" dans le passé en fonction de la mémoire associé à chaque modèle. Un processus est marqué comme malicieux si un seul modèle "process-centric" déclare pour K intervalles consécutifs que les données sont effectivement malicieuse. Les auteurs disposent ainsi de modèles prenant en compte : (1) attaque mono-processus et multi-processus, (2) rapidité de détection avec les intervalles fixes et (3) l'histoire passé de chaque processus à différents degrés. Les modèles sont implémentés avec des forêts d'arbres décisionnels de 100 arbres. Chaque arbre produit en sortie : -1 pour bénin ou +1 pour malicieux. La sortie cumulée d'une forêt est donc comprise entre -100 et +100. Si jamais le score obtenu est 0, alors le processus est marqué comme suspect. Un modèle "system-centric" intervient alors afin de prendre une décision. Si un processus obtient deux fois un score de 0 alors celui-ci est marqué comme malicieux. La FIGURE 2.5 présente un exemple de l'utilisation de modèles incrémentals pour la détection.

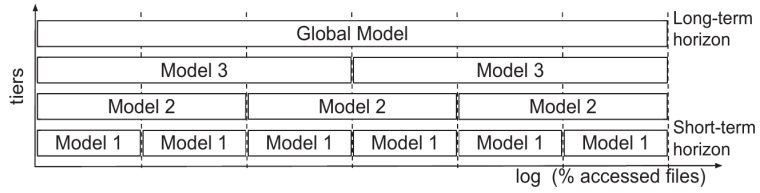

FIGURE 2.5 – (Continella *et al.* [71] [10]) Un exemple de l'utilisation de modèles incrémentals. À chaque intervalle, on vérifie le comportement du processus surveillé pour chaque modèle.

Les échantillons de la famille *Citroni* ont un comportement particulier qui biaise les caractéristiques utilisées par les modèles. En effet, ceux-ci copient le fichier original dans un fichier temporaire (i.e., toujours le même nom), le chiffre puis écrasent le fichier original. SHIELDFS observe ainsi de nombreux renommages puis de nombreuses opérations de lecture et écriture sur un seul et même fichier. Pour contrer cela, SHIELDFS contrôle si un fichier est lu ou écrit peu de temps après avoir été déplacé ou renommé. Il s'agit là d'une rustine sans coût supplémentaire d'après les auteurs. Cela illustre néanmoins le fait que des techniques d'évasions peu évoluées

10. Reproduit avec la permission de *Association for Computing Machinery, Inc (ACM)*

peuvent rendre inefficaces les algorithmes d'apprentissage automatique utilisés ici.

Les caractéristiques prises en compte par les modèles d'apprentissage doivent être normalisées en fonction de particularités propres à chaque système (e.g., nombre total de fichiers). Pour ce faire, il est nécessaire de mettre à jour certaines de ces valeurs car elles évoluent en fonction du temps. Les auteurs choisissent de mettre à jour les facteurs normalisés d'un système en temps réel. Ils indiquent néanmoins qu'une modification en temps réel peut permettre à un logiciel de rançon de manipuler ces facteurs et ainsi de ne pas être détecté. Avec une mise à jour quotidienne, le logiciel de rançon peut toujours manipuler les facteurs mais doit alors attendre le lendemain pour ne pas être détecté. Cette méthode est plus résiliente mais déclenche un faux positif comme nous le ferons pas la suite, ce qui explique qu'elle ne soit pas retenue.

Le module en charge de la restauration des fichiers constitue la seconde partie de ShieldFS après le module de détection. Ainsi à chaque nouveau processus est associé le statut "unknown". Quand un tel processus va pour la première fois écrire ou supprimer des fichiers, ceux-ci vont être copiés dans une zone accessible seulement par ShieldFS. Une fois le premier intervalle échu, le processus bascule dans un des états suivants : bénin, suspect ou malicieux. Si celui-ci est bénin alors on supprime les fichiers sauvegardés qui lui sont associés. S'il est malicieux, c'est-à-dire au moins K scores malicieux consécutifs pour un des modèles, ShieldFS suspend le processus puis restaurer les fichiers. En cas de match nul, le modèle global désigne le modèle indécis pour l'intervalle considéré comme bénin ou malicieux. Cette précision n'est pas apportée dans le papier, mais elle me semble intuitive. Une fois qu'un processus bascule dans l'état malicieux (e.g., premier "tick") alors tous les fichiers qu'il modifie sont sauvegardés. ShieldFS garde en mémoire les opérations qui conduisent à la modification des fichiers. Ce qui permet de revenir sur les opérations malicieuses. La portion du disque protégée qui assure la sauvegarde des fichiers peut être vue comme un cache. En effet, le même fichier n'est pas copié X fois. Les auteurs gardent dans le cache une copie pendant au plus T heures. D'après les expérimentations (i.e., données des utilisateurs), ils évaluent ce T entre 1 et 4 heures. Il s'agit d'un compromis entre performance et efficacité du module de restauration. De plus, les données qui n'ont aucune valeur pour un utilisateur ne sont pas sauvegardées. Les auteurs prennent pour exemple les dossiers temporaires et de caches propres à chaque application. Ces répertoires et les fichiers qu'ils contiennent font ainsi partie d'une liste blanche. Ils sont fréquemment utilisés ce qui permet à la solution d'optimiser ses performances. Pour éviter un logiciel de rançon d'exploiter ce mécanisme. Tout nouveau processus qui n'a jamais accédé à un dossier en liste blanche est considéré comme suspect, s'il tente de déplacer des fichiers dans un de ces dossiers. Les fichiers ainsi déplacés sont de manière préventive copiés dans le disque sécurisé.

Les auteurs discutent du Volume Shadow Copy Service de Windows. Celui-ci est facilement contournable, les logiciels de rançon procèdent à sa destruction systématique avant de commencer à chiffrer. Enfin, il ne suit pas le même objectif que ShieldFS, c'est-à-dire sauvegarder des copies récentes des fichiers à court terme.

Un troisième composant est présent dans ShieldFS, celui-ci à pour objectif de détecter des primitives cryptographiques et plus précisément d'en extraire les clés. Les auteurs ciblent les algorithmes de chiffrement symétriques par bloc. Ils conviennent aux besoins des logiciels de rançon. Pour des raisons d'optimisations il est possible de précalculer une structure que l'on appelle : le

"key schedule". Les auteurs observent que la plupart des librairies (e.g., OpenSSL) et des logiciels de rançon précalculent bien le "key schedule". Le principal effet de bord de ce précalcul est que la clé secrète est disséminée dans cette structure de données. Il est donc impératif que celle-ci réside uniquement en mémoire durant le chiffrement. Les auteurs implémentent CryptoFinder, celui-ci scanne la mémoire des processus à la recherche de cette structure. De plus, de part leur nature les algorithmes à clé secrète sont publiques (i.e., principe de Kerckhoffs). La structure associée à un "key schedule" est donc connue. Il est alors très peu probable de trouver une suite d'octets qui correspondent à cette structure alors qu'en fait c'est un faux positif. Un faux négatif peut arriver si cette structure n'est pas contiguë en mémoire (i.e., plusieurs pages). La taille du "key schedule" étant faible, le chevauchement sur deux pages est peu probable. CryptoFinder est plus anecdotique qu'autre chose. Il n'est pas assez générique pour assurer à lui seul la restauration des fichiers. Il est employé dans un cas de figure précis par le module de détection. C'est-à-dire lorsque qu'un processus vient de basculer dans l'état malicieux mais pas encore pour K "ticks" (i.e., définitif). CryptoFinder recherche alors la présence d'un éventuel algorithme de chiffrement symétrique dans sa mémoire. S'il en trouve trace, le processus est marqué comme malicieux de manière définitive et la restauration intervient. Sinon, rien ne se passe et ShieldFS attend les $K - 1$ scores malicieux consécutifs suivants pour intervenir. Les auteurs insistent sur le fait que CryptoFinder est optionnel. Le module de détection est tout à fait capable de fonctionner sans celui-ci. Les auteurs ciblent l'algorithme Advanced Encryption Standard (AES) pour cette preuve de concept. La figure 2.6 présente l'architecture de ShieldFS, notamment les trois composants qui forment cette contre-mesure.

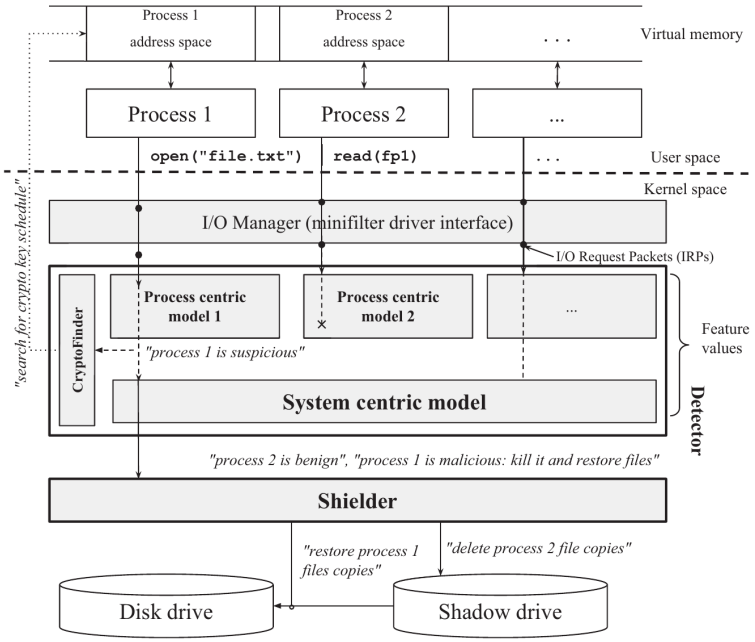

Figure 2.6 – (Continella *et al.* [71] [11]) Architecture de ShieldFS.

11. Reproduit avec la permission de *Association for Computing Machinery, Inc (ACM)*

Les expérimentations ont lieu avec des machines virtuelles, dont le contenu est représentatif des données d'un utilisateur. ShieldFS est bien entendu un pilote de type "file system minifilter driver". Les auteurs vérifient que la phase d'entraînement n'a pas engendré de sur-apprentissage. Pour cela, les auteurs procèdent à trois tests de validation croisés différents. Les résultats (e.g., taux de détection) sont basées sur la granularité d'un processus.

Le premier, divise l'ensemble d'entraînement des logiciels de rançon en 10 échantillons (i.e., 10-fold cross validation) de manière aléatoire. Cet ensemble de tests contient donc au total 383 logiciels de rançon. Dans ce paragraphe les auteurs considèrent un processus comme malicieux seulement s'il déclenche un score suspect une seule fois (i.e., -100). Il n'est pas question d'attendre K intervalles consécutifs. Ce même K n'étant pas encore déterminé. Les résultats sont excellents. Un taux de faux positifs compris entre 0.0 et 0.015% est obtenu. Le taux de détection quant à lui est d'environ 90% pour 1k IRPs interceptées avec les modèles "process-centric". Il progresse ensuite lentement avec l'augmentation du nombre d'IRPs collectées. Les modèles orientés processus ont besoin d'au moins 1k IRPs. Avant cela le système de détection ne réagit pas. Il faut 10k IRPs pour que le modèle orienté système atteigne un taux de détection de 90%. Celui-ci est plus lent pour détecter un comportement anormal. Les auteurs discutent aussi du taux de détection obtenu en utilisant une approche non incrémentale. C'est-à-dire sans utiliser les modèles représentant X% des fichiers touchés. Ils testent donc les modèles basés sur l'ensemble des données d'apprentissage, seule la mémoire de ces modèles varient encore (i.e., N "ticks" dans le passé). Il faut alors 100k IRPs pour qu'ils commencent à réagir à une attaque. L'approche incrémentale est donc un bon choix et permet de réagir bien plus vite à une attaque, ce qui conforte les auteurs.

Le second test a pour objectif de démontrer l'indépendance des résultats obtenus avec les données d'entraînements et de tests. Ce test de validation croisé s'intéresse aux donnés utilisateurs. Le test précédent étant focalisé sur les logiciels de rançon. Les auteurs sélectionnent ainsi les données d'une des 11 machines utilisateurs et les retirent de l'ensemble d'entraînement. Les données ainsi retirées sont ensuite utilisées comme un ensemble de test (i.e., one machine off cross validation). Cette procédure est répétée pour chacun des 11 jeux de données utilisateurs. Les résultats montrent que le fait de retirer un jeu de données de l'apprentissage puis de le tester comme ensemble de test n'a pas d'impact sur le taux de faux positif. Il n'y a donc pas de grande dépendance entre les données d'entraînements et de tests. Durant cette phase de validation croisée, les auteurs découvrent deux cas de faux positifs. Le premier est déclenché par *explorer.exe*. Ce processus accède à un nombre très important de fichiers sur la machine de l'utilisateur 1. Ce problème motive la mise à jour des facteurs de normalisation en temps réel. Malheureusement de l'aveu même des auteurs, cela peut être exploité par les logiciels de rançon pour manipuler les facteurs de normalisations. Il s'agit d'un compromis inhérent au fonctionnement de ShieldFS. Le second faux positif est provoqué par *Visual Studio*. Parmi les 32 sessions de *Visual Studio* enregistrées, seulement une est détectée comme malicieuse. La session malicieuse écrit 175 fichiers avec une grande entropie. Les caractéristiques restantes ne semblent pas capables d'éliminer ce faux positif.

Le dernier test détermine de manière empirique la valeur de K. C'est-à-dire le nombre consécutif de scores malicieux pour un processus qu'il faut obtenir d'un modèle pour marquer ce même processus comme malicieux. Les auteurs font varier K de 1 à 6 sur le jeu de données des logiciels

de rançon et des utilisateurs en divisant chacun de ces deux ensembles en 10 ensemble de tests (i.e., 10-fold cross validation). Pour $K = 3$, le taux de détection est de 100% pour un taux de faux positifs de 0.038%. Malheureusement les auteurs ne rapportent pas à combien de processus 0.038% de faux positifs correspond. De plus, les auteurs ne disent pas pour quelle mémoire N ils obtiennent de tels résultats ni au bout de combien d'intervalles. La connaissance de ces paramètres aurait été intéressant pour apprécier le fonctionnement de la solution. Les auteurs indiquent que le taux de détection de 100% est assuré en interceptant 67k IRPs par processus. Tout du moins c'est la compréhension que j'ai d'un de leur tableau. Ce qui tranche nettement avec les 1k vue précédemment, c'est-à-dire lorsqu'il n'est pas nécessaire d'attendre K sorties consécutives malicieuses. La taille des intervalles qui augmentent de manière exponentielle ainsi que le souhait de diminuer les faux positifs font qu'il est nécessaire d'engranger davantage de données. Pour rappel, le module de détection ne s'exécute qu'une fois un intervalle échu. C'est-à-dire pour un pourcentage de fichiers touchés par un processus. L'application malicieuse ne met sans doute que quelques secondes pour générer 67k IRPs, mais à combien de fichiers cela correspond-il ?

Une fois les tests de validation croisé terminés, les auteurs procèdent à deux expériences supplémentaires. La première expérience consiste à choisir trois logiciels de rançon puis à infecter trois machines distinctes utilisées par de "vrai" utilisateurs. Les auteurs rapportent que les trois échantillons ont été détectés sans que les utilisateurs ne perdent le moindre fichier. Le module de restauration est intervenu. Malheureusement, nous ne disposons pas de davantage de détails (e.g., temps de détection, nombre de fichiers chiffrés). La seconde expérience est plus importante. Les auteurs mettent en place un environnement d'analyse avec des machines virtuelles dans le but de tester ShieldFS sur de nouveaux logiciels de rançon. Chaque analyse dure 60 minutes. 305 échantillons sont ainsi récupérés et aucun n'était présent dans le corpus d'entraînement. De plus, 7 nouvelles familles sont présentes dans ce corpus. ShieldFS détecte 97.7% des logiciels de rançon et ce sans aucun faux positif. L'ensemble des fichiers chiffrés ont été restaurés avec succès. Un seul cas de processus marqué comme suspect a été déclaré par les modèles "process-centric". Le modèle orienté système n'a donc été invoqué qu'une seule fois. De plus, CryptoFinder qui ne devait être qu'une pièce de moindre importance contribue à la détection de 69.3% des échantillons. En effet, il permet de marquer un processus comme définitivement malicieux dès que celui-ci obtient un premier score malicieux. Il n'est pas nécessaire d'attendre K scores malicieux consécutifs. Les 7 faux négatifs ont un comportement bien particulier. Ces échantillons ne chiffrent que 30 fichiers chacun alors que l'analyse dure 60 minutes. Ce comportement est peut-être du à l'environnement d'analyse (i.e., machine virtuelle) ou à une volonté de chiffrer lentement. Ils ne sont pas détectés mais le module de restauration de ShieldFS est capable de restaurer les fichiers chiffrés. En effet, à son lancement tout nouveau processus est dans l'état "unknown" et les fichiers qu'il touche sont systématiquement sauvegardés. Pour résumer cette campagne de test, 100% des échantillons ont un effet nul sur le système. Le modèle orienté système n'est finalement pas très utile (i.e., une invocation). Les 7 faux négatifs n'exploitent que partiellement les faiblesses de ShieldFS, volontairement ou involontairement. Ils pourraient procéder de manière similaire mais en s'injectant dans un processus marqué comme bénin. Dans ce cas, ShieldFS est inefficace.

Les auteurs concluent les expérimentations avec une suite de tests pour apprécier l'impact sur le système en terme de performance de ShieldFS. Le premier test a pour objectif de quan-

tifier avec une bonne approximation le coût supplémentaire que ShieldFS introduit pour un utilisateur lors de ses activités courantes. Pour cela, les auteurs collectent de nouvelles données utilisateurs pendant un mois. À partir de ces données ils construisent une séquence d'IRPs qui s'étend sur 6 heures. Puis ils font correspondre à chaque IRP la fonction correspondante (e.g., *FileRead*). Ils estiment ensuite le surcoût de ShieldFS grâce à cette abstraction sans donner davantage de détails. À combien de millisecondes ou de microsecondes les auteurs estiment le coût supplémentaire lors de la lecture d'un fichier ? Le résultat est le suivant : l'impact supplémentaire est estimé en moyenne à 26%. De plus, ils indiquent qu'un utilisateur je cite : "*barely perceived it while using a machine protected by* ShieldFS".

Des tests temps réels sont également réalisés. Les auteurs simulent 3 séries d'opérations différentes (e.g., ouverture puis lecture) sur des fichiers de tailles différents (i.e., de 1Ko jusqu'à 128 Mo). Les auteurs comparent également ces opérations avec et sans le module de restauration. Chaque série est répétée 100 fois. Ces tests se déroulent sur un Windows 10 équipé d'un disque dur. Les performances chutent drastiquement. Nous observons figure 2.7, un surcoût compris entre 180% et 380% quand les fichiers sont sauvegardés. Sans la sauvegarde, le surcoût est compris entre 30% et 90%. Le temps nécessaire pour réaliser des opérations courantes avec sauvegarde est au minimum presque doublé par rapport au temps de référence sans ShieldFS.

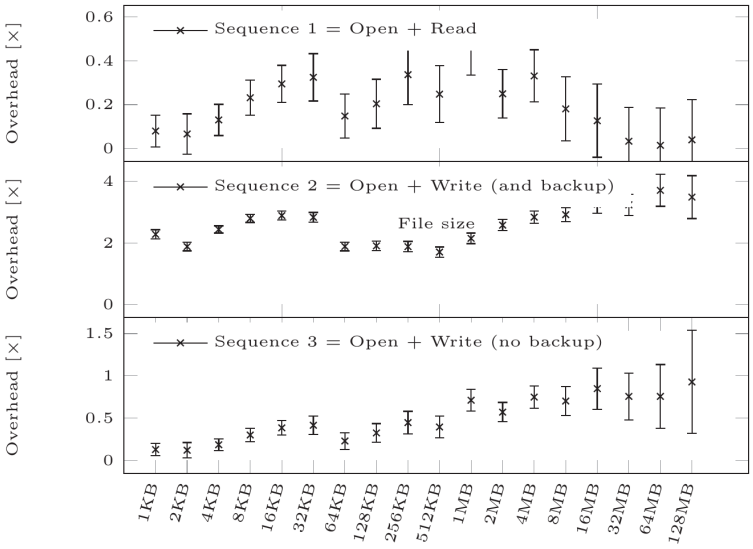

Figure 2.7 – (Continella *et al.* [71] [12]) Série de 3 tests temps réels et surcoût moyen de ShieldFS.

Même sans le module de sauvegarde les résultats sont trop mauvais pour espérer déployer la solution. De plus, cette suite de tests est une nouvelle fois non reproductible. Un dernier point, les auteurs utilisent un disque dur. Les disques durs ont des performances bien moindre que les solid-state drives. Avec un SSD les pertes de performance auraient été beaucoup plus importante. Les auteurs évaluent également l'espace occupé par le module de restauration. Ils font varier le paramètre T de 1 à 4 heures. Celui-ci a une influence minime sur les performances. Les auteurs choisissent $T = 3$. Avec un tel paramètre, durant les expérimentations c'est au

12. Reproduit avec la permission de *Association for Computing Machinery, Inc (ACM)*

maximum 14.73 Go et en moyenne 500 Mo qui ont été occupés.

La dernière section du papier discute des limitations. Une prise de décision est déclenchée à chaque intervalle (i.e., "tick). Seulement les intervalles ne sont pas fonction du temps mais du nombre de fichiers touchés par un processus. Il est donc possible pour un logiciel de rançon de chiffrer peu de fichiers sans ce faire détecter. Par exemple, en lançant de multiple instances qui se contentent de chiffrer 10 fichiers. Dans ce cas, il est aussi nécessaire d'attendre T heures que le système de sauvegarde supprime les quelques fichiers touchés. Il est difficile pour un logiciel de rançon de réaliser cette attaque sans la connaissance de T, et surtout l'utilisateur risque de s'en apercevoir. Un logiciel de rançon peut aussi être plus malin en s'injectant dans un processus bénin ayant déjà accédé à de nombreux fichiers. La taille des intervalles augmentent de manière exponentielle, il peut donc chiffrer de nombreux fichiers avant d'être détecté au prochain "tick". De plus, le processus étant bénin, aucune sauvegarde des fichiers ne saura réalisée. Même s'il est optionnel, le module CryptoFinder a contribué à détecter pas moins de 69% des échantillons. L'utilisation d'instructions spécifiques comme AES-NI [19] rend ce module inefficace. On peut donc se demander dans quelle mesure cela affecterait la détection des logiciels de rançon. De plus, le module de restauration peut être volontairement trompé en faisant une attaque par déni de service. Une fois celui-ci plein, le logiciel de rançon passe à l'attaque. Au tout début du papier lors de la phase d'entraînement les auteurs distinguent les opérations sur les fichiers utilisateurs et les fichiers systèmes. Ils entraînent ainsi deux ensembles de modèles : utilisateur et système (i.e., pas au sens "system-centric"). Ils n'en font plus mention par la suite. Enfin, l'utilisation de l'apprentissage automatique supervisé pour détecter des logiciels de rançon n'est pas parfait. Comme nous l'avons vu avec une famille *Citroni*, de simple modifications de comportements peuvent suffire à berner les modèles. Obligeant les auteurs à davantage de surveillance et donc à faire baisser les performances du système. La dernière limitation n'est pas des moindre est le coût pour le système. ShieldFS ne peut être déployé sur une machine utilisateur. D'autre part, certains utilisateurs peuvent avoir une utilisation très particulière de leur poste (e.g., application métier). Il est alors nécessaire de mettre au point de nouveaux modèles ou alors faire de l'apprentissage par renforcement.

En conclusion, Continella *et al.* [71] présentent une contre-mesure, ShieldFS, qui annonce un taux de détection de 100%. Le nombre de faux positifs est plus difficile à apprécier. En effet, 0.038% des processus déclenchent une fausse alerte lors de la phase de test. Seulement les auteurs ne disent pas à combien de processus cela correspond. Il s'agit de la première implémentation significative d'un système de détection des logiciels de rançon temps réel basé sur l'apprentissage automatique supervisé. ShieldFS ne peut être déployé en raison de son coût pour le système. En effet, les auteurs mesurent jusqu'à 380% de latence supplémentaire. ShieldFS contient également un module de restauration pro-actif, ce qui étend ces capacités par rapport à CryptoDrop de Scaife *et al.* [103]. Les auteurs [71, 103] surveillent toujours un nombre important de paramètres du système de fichiers. Nous verrons si les contributions suivantes tentent d'adresser le problème des performances en étant plus minimaliste.

Redemption : Real-Time Protection Against Ransomware at End-Hosts [88]

En 2017, Kharraz *et al.* [88] présentent après deux contributions dans le domaine [89, 87] leur première contre-mesure temps réel appelée REDEMPTION. Cette contribution est centrée autour de quatre axes : (1) détection basée sur les I/Os du système de fichiers, (2) faible impact sur le système et ce même pendant une attaque, (3) garantir qu'aucun fichier utilisateur ne sera perdu et (4) une expérience utilisateur. Le premier axe est classique et permet de détecter des échantillons encore inconnus, car la sémantique des logiciels de rançon est connue. Les vraies nouveautés de cette contribution sont indéniablement le faible impact sur le système et l'expérience utilisateur. Le troisième axe quant à lui réduit à néant le comportement malicieux des logiciels de rançon.

REDEMPTION a deux composants : (1) un pilote de type "file system minifilter driver" et (2) un programme utilisateur. Le premier intercepte les I/Os sur le système de fichiers avec la granularité d'un processus et assure la résilience du système en cas d'attaque. La résilience est assurée en gardant en mémoire les opérations suspectes sur les fichiers utilisateurs mais aussi en ne les répercutant pas directement sur ces mêmes fichiers. Ce mécanisme de mise en cache est totalement transparent pour les utilisateurs. Le second composant qui se trouve dans l'espace utilisateur est en charge de surveiller en temps réel le comportement de chaque processus et d'avertir l'utilisateur en cas de suspicion. En fait, REDEMPTION redirige toutes les requêtes en écriture sur les fichiers utilisateurs. Le fichier original est copié dans la zone protégée et les modifications appliquées. REDEMPTION répercute périodiquement les modifications des fichiers tampon sur le disque afin d'assurer la cohérence des données. Les auteurs ne disent pas à quelle fréquence et ne discutent pas non plus du temps pendant lequel la zone protégée sauvegarde un fichier. L'architecture de REDEMPTION est visible FIGURE 2.8.

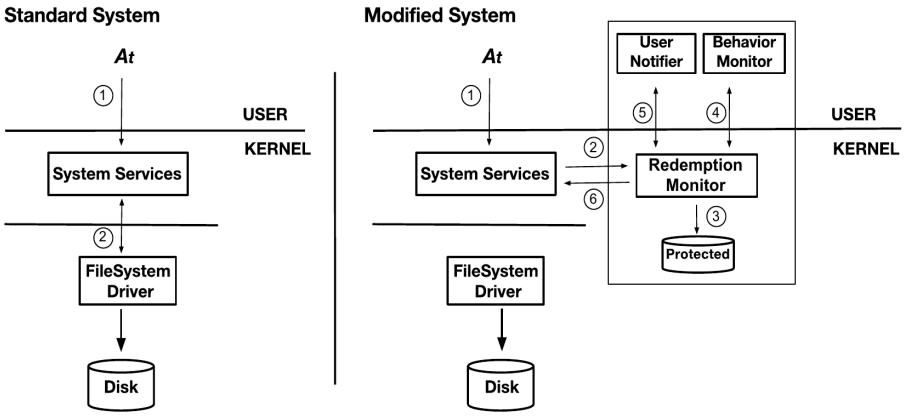

FIGURE 2.8 – (Kharraz *et al.* [88] [13]) Architecture de REDEMPTION.

Les applications de l'espace utilisateur qui souhaitent accéder à des fichiers utilisateurs sont mises sous surveillance. On assigne un score à ces applications, celui-ci représente le risque que le processus exhibe un comportement malveillant. Ce score est calculé à partir de 6 indicateurs de compromission divisés en deux ensembles : (F_1) modification du contenu des fichiers et (F_2)

13. Reproduit avec la permission de *Springer International Publishing*

comportement général sur les fichiers (i.e., "cross-file behavior"). Le premier ensemble est composé des trois indicateurs suivants :

1. ratio de l'entropie de Shannon pour un même fichier ouvert en lecture puis écriture,
2. réécriture d'un fichier, plus de parties différentes sont touchées et plus le score est malicieux,
3. suppression d'un fichier.

Le second ensemble contient les indicateurs suivants :

4. parcourt des répertoires et énumération des fichiers,
5. changement de type des fichiers,
6. fréquence avec laquelle un processus accède à deux fichiers distinct consécutivement.

Les auteurs ne pensent pas que les 6 indicateurs contribuent de manière égale à la détection des logiciels de rançon. Le score est fonction de ces indicateurs, et celui-ci leur attribue un poids. Les poids sont déterminés par des expérimentations préliminaires sur des traces utilisateurs et des traces de logiciels de rançon. Si l'on ne considère que les indicateurs F_1, un taux de faux positifs de 5.9% est obtenu. Une des raisons est que les applications de compression si elles sont configurées pour supprimer le fichier original déclenchent une alerte. De même que la manipulation de fichiers qui possédent avec une entropie élevée. L'ajout des indicateurs F_2 permet de faire chuter le taux de faux positifs à 0.5% tout en fournissant un taux de détection de 100%. Pour chaque processus X, un score S lui est assigné à chaque fois qu'il accède à un fichier utilisateur. Si le score S dépasse une valeur de seuil prédéfinie α alors le processus est suspendu et l'utilisateur est notifié afin de confirmer ou infirmer les modifications en cours.

La fonction de score malicieuse (i.e., MSC) au moment t est la suivante :

$$MSC(r) \;=\; \frac{\sum_{i=1}^{k} w_i \;\times\; r_i}{\sum_{i=1}^{k} w_i} \tag{2.5}$$

Pour l'indicateur i on associe le poids w_i et le score r_i. Les auteurs font remarquer que même l'utilisation combinée de tous les indicateurs ne permet pas d'écarter certain faux positifs. En particulier, les applications de chiffrement et de suppression sécurisée. Celles-ci ont un comportement trop similaire aux logiciels de rançon.

Les auteurs implémentent REDEMPTION sur la plateforme Windows, celle-ci étant la plus ciblée. Les mécanismes de défenses sont génériques et peuvent être transposés à n'importe quel autre système d'exploitation. REDEMPTION surveille l'ensemble des I/Os via son pilote. De telle sorte qu'il est impossible pour un logiciel de rançon d'accéder à des fichiers sans notifier le pilote. Le programme utilisateur est un service particulier de type anti-logiciels malveillants (i.e., "anti-malware user mode services") introduit par Microsoft pour protéger les applications de sécurité dans l'espace utilisateur. En effet, le composant qui communique avec un pilote est le point le plus facile à attaquer pour un logiciel malveillant. Microsoft ajoute donc des mécanismes de sécurité supplémentaires à partir de Windows 8.1. Ces mécanismes sont : (1) vérification de l'intégrité du code, (2) authentification du code (i.e., signature), (3) protection contre l'injection et (4)

protection contre l'élévation de privilège. REDEMPTION est implémenté sur Windows 7, sauf erreur de ma part le service anti-logiciels malveillants n'est pas disponible pour cette version.

Avant de passer à la phase d'expérimentation, les auteurs collectent des données provenant de 5 machines utilisateurs. Ils récupèrent ainsi plus de 230 *Go* de données saines contenant les interactions de processus bénins avec le système de fichiers. L'objectif est d'avoir suffisamment de données comme base d'entraînement pour le "modèle" mais aussi de représenter des cas d'usages réels. Les auteurs disposent également d'un corpus de 1174 logiciels de rançon actifs répartis en 29 familles. Pour chacun des échantillons, une trace des opérations sur le système de fichiers est créée lors de son analyse. Les auteurs ne disent pas comment les données sont récupérées, sans doute par manque de place. On peut imaginer qu'ils aient utilisé comme dans leur précédente contribution [87], un environnement virtualisé avec la *Cuckoo Sandbox*.

L'objectif de la première expérimentation sur les données d'entraînement est de déterminer la valeur de seuil α qui permet de distinguer les logiciels bénins des malicieux. Bien sûr, celle-ci doit être choisie de telle sorte qu'elle ne déclenche que très peu de faux positifs. Les auteurs procèdent à un échantillonnage des 504 échantillons d'entraînement et des traces saines (i.e., "10-fold cross validation). Ils ne disent pas combien de gigaoctet provenant des données utilisateurs sont utilisés pour cette phase d'entraînement. Les scores des applications bénignes sont significativement plus faible que ceux des logiciels malveillants. La médiane des scores maximum pour 22 applications bénignes est de 0.0885 alors qu'elle est de 0.73 pour les logiciels de rançon. Les applications de chiffrement tel que *AESCrypt* ont un score élevé, ce qui déclenche des faux positifs. Malheureusement les faux positifs sont inévitables compte tenu de la nature de ces programmes. Les auteurs fixent la valeur de seuil ou score malicieux $\alpha = 0.12$. Avec un tel paramétrage, REDEMPTION obtient un taux de détection de 100% et un taux de faux positifs de 0.5%. On ne sait pas à combien de processus suspects correspondent 0.5% de faux positifs ni à quelle fréquence on demande à l'utilisateur d'intervenir quotidiennement en moyenne. REDEMPTION détecte très rapidement une attaque, en moyenne il a besoin d'observer le chiffrement de 4 fichiers utilisateur. Les auteurs évaluent également la taille de la zone protégée à 7 *Go* sans donner plus de détails. Par ailleurs, on ne sait pas pendant combien de temps les fichiers sont gardés dans le cache.

Une seconde expérimentation est menée sur 677 échantillons inconnus par le modèle, c'est-à-dire non-présent lors de l'entraînement, et un second jeu de données utilisateur. Les auteurs ont vérifié manuellement les résultats. REDEMPTION détecte 100% des logiciels de rançon pour un taux de faux positifs de 0.8%. 5 fichiers sont nécessaires en moyenne pour que REDEMPTION détecte une attaque. Les auteurs concluent que REDEMPTION est capable de détecter des échantillons inconnus tout en conservant un nombre de fichiers attaqués (i.e., mis en cache) sensiblement identique (i.e., 4 puis 5). Il faut relativiser quelque peu les propos, car dans les deux ensembles, c'est-à-dire d'entraînement et de test, figurent des échantillons de la même famille (i.e., 12 familles phase d'entraînement puis 29 phase de test). Au sein d'une même famille, le comportement sur le système de fichiers est sensiblement identique, ce sont les algorithmes cryptographiques qui peuvent changer (e.g., une fois cryptanalysé).

L'évaluation des performances de REDEMPTION sur le système de fichiers est réalisée avec un outil de test standard : IOzone [20]. Le test suivant est effectué : (1) génération de 100×512 Mo fichiers puis accès en écriture, réécriture et lecture (2) génération de 50k fichiers de 1 Mo puis

accès en lecture. Ce test est répété 10 fois. La FIGURE 2.9 présente ces résultats. Un surcoût de 2.8% et 3.4% en moyenne est induit par REDEMPTION pour les opérations de lecture et d'écriture. Le coût est plus élevé pour le système de fichiers avec les opérations de réécriture et de création de fichiers, respectivement 9% et 7%. L'explication est simple, lorsque le système de fichiers est soumis à une activité intensive, par exemple la création de 50k fichiers (i.e., accès en écriture), REDEMPTION doit copier les fichiers originaux dans la zone protégée. Deux questions importantes se posent alors, combien de fichiers le pilote peut-il maintenir dans la zone protégée et pour combien de temps ? En effet, cela a une répercussion directe sur les performances mais aussi sur la capacité de REDEMPTION à assurer qu'aucun fichier ne soit perdu. Les auteurs répondent de manière évasive en affirmant : "*the system can maintain several files*". D'autre part, dans certains scénarios de l'aveu même des auteurs, l'application utilisateur doit pouvoir accéder immédiatement à des données écrites quelques millisecondes plus tôt. Les auteurs pensent par exemple aux bases de données. Dans ce cas, il faut également rediriger les requêtes en lecture vers la zone protégée. Les auteurs font un test de performance sur le cache en lecture seulement. Ils obtiennent un surcoût de 3.1% quand 100 fichiers de taille médiane 17.4 Mo sont lus. Trop de paramètres ne sont pas déterminés et/ou évalués correctement. On passe d'une solution qui ne redirige que les requêtes en écriture vers le cache à une solution qui redirige également les requêtes en lecture. La taille du cache est arbitraire (i.e., 7 Go) et la durée de mise en cache des modifications avant répercussion (i.e., commit) sur le système de fichiers est indéterminée.

Operation	Original Performance	Redemption Performance	Overhead(%)
Write	112,456.25 KB/s	110094.67KB/s	3.4%
Rewrite	68,457.57 KB/s	62501.76 KB/s	8.7%
Read	114,124.78 KB/s	112070.53 KB/s	2.8%
Create	12,785 files/s	11,852 files/s	7.3%

FIGURE 2.9 – (Kharraz *et al.* [88] [14]) Les performances de REDEMPTION pour différents scénarios sur le système de fichiers en utilisant IOzone. Il n'est pas dit si les auteurs utilisent un disque dur ou un SSD.

Une évaluation de la latence perçue par un utilisateur est également effectuée. Afin d'interagir et d'automatiser les tests avec des applications graphiques, les auteurs utilisent un outil appelé *AutoIt* [2]. Une dizaine d'applications sont ainsi testées une dizaine de fois. Les résultats sont visibles FIGURE 2.10. La durée des tests est en moyenne 2.6% plus longue et dans le pire cas en dessous de 6%. Les auteurs concluent en exprimant leur satisfaction de voir que leur solution peut-être déployée sur la machine d'un utilisateur lambda.

REDEMPTION surveille l'ensemble des I/Os provenant de l'espace utilisateur, ce qui fait environ une dizaine d'IRPs. Le fait d'enregistrer une fonction pour chacune de ces opérations (i.e., "callback function") a un coût, ainsi que les traitements propre à REDEMPTION. Les auteurs ne discutent pas de la communication entre le pilote et l'application sécurisée dans l'espace utilisateur. Pour associer un score à chaque processus cette dernière doit recevoir en continu des

14. Reproduit avec la permission de *Springer International Publishing*

Application	Original (s)	Redemption (s)	Overhead (%)
AESCrypt	165.55	173.28	4.67%
AxCrypt	182.4	191.72	5.11%
Chrome	66.19	67.02	1.25%
IE	68.58	69.73	1.67%
Media Player	118.2	118.78	0.49%
MS Paint	134.5	138.91	3.28%
MS Word	182.17	187.84	3.11%
SDelete	219.4	231.0	5.29%
Vera Crypt	187.5	196.46	4.78%
Winzip	139.7	141.39	1.21%
WinRAR	160.8	163.12	1.44%
zip	127.8	129.32	1.19%
Average	-	-	2.6%

FIGURE 2.10 – (Kharraz *et al.* [88] [15]) Surcoût de REDEMPTION sur des applications bénignes.

informations du pilote. L'évaluation des performances a t-elle été réalisée avec le programme utilisateur ? De plus, le fait de rediriger des requêtes dans un cache peut entraîner l'incohérence des données ou le plantage d'une application. L'ingénierie du système est très complexe, une approche plus simple aurait été souhaitable. Enfin, les auteurs ne disent pas si les expérimentations ont eu lieu avec un disque dur ou un SSD.

La dernière contribution du papier est consacrée à une expérimentation utilisateur. Le premier objectif est de voir si les utilisateurs sont capables d'interpréter correctement les alertes remontées par REDEMPTION. Le second est de démontrer que la surveillance de REDEMPTION est transparente pour un utilisateur, c'est-à-dire sans perte de performance perceptible. Cette expérience mobilise 28 participants de différentes sensibilités. De plus, ceux-ci ne sont pas tenus au courant du "vrai" déroulement de l'expérience. Les auteurs ont préalablement reçu les autorisations nécessaires de la part de leur institution. Les expériences se déroulent sur deux ordinateurs portables équipés chacun d'une machine virtuelle Windows 7 avec REDEMPTION d'activé et une connexion Internet. Les participants doivent ensuite réaliser séquentiellement trois tâches. La tâche (A) est de manipuler des outils de traitement de texte sur la machine de test. Un membre de l'expérimentation demande ensuite au participant de comparer cette expérience avec sa pratique quotidienne de ces outils sur sa machine. L'utilisateur mesure donc l'interactivité du système grâce à l'échelle de Likert. La tâche (B) est de chiffrer volontairement un répertoire. Cette action déclenche la fenêtre d'alerte de REDEMPTION, suspend l'opération et demande à l'utilisateur de confirmer ou infirmer l'action. Il est ensuite demandé au participant d'argumenter son choix. La tâche (C) demande au participant de naviguer sur Internet pour effectuer une recherche. Un logiciel de rançon est exécuté pendant cette tâche et déclenche ainsi la fenêtre d'alerte. Comme précédemment, l'utilisateur enregistre sa réponse tout en justifiant son choix. Les résultats montrent qu'aucun des 28 participants ne perçoit un quelconque ralentissement lors de la tâche (A). REDEMPTION est donc transparent pour les utilisateurs. Concernant la tâche (B), 26 participants confirment l'action légitime qu'ils ont eux même intentée. Deux d'entre eux seulement bloquent l'action. Pour la troisième tâche (C), tous les participants ont

15. Reproduit avec la permission de *Springer International Publishing*

correctement bloqué le logiciel de rançon. Les utilisateurs sont donc capables de reconnaître des actions malicieuses sur le système grâce aux indicateurs de REDEMPTION. Pour confirmer cette hypothèse les auteurs font un test de Student. L'hypothèse nulle étant que REDEMPTION ne fournit aucune aide pour identifier un processus malicieux. L'hypothèse nulle est rejetée par le test de Student car la p-value est bien inférieure au degré de signification $\alpha = 0.01$ fixé. Les auteurs acceptent donc l'hypothèse alternative qui est : REDEMPTION avec ses indicateurs assiste un utilisateur pour détecter un comportement suspect. On regrette que les auteurs aient omis certains détails (e.g., durée des tâches). Le fait de faire passer trois tâches séquentiellement aux participants peut introduire un biais, de même que leur ordre, qui ici n'est pas donné. La fenêtre d'alerte n'est pas fourni.

Finalement, les auteurs discutent des limitations. Il est possible pour un logiciel de rançon de faire de l'ingénierie sociale en frustrant l'utilisateur avec de fausses alertes usurpant l'identité d'un programme légitime. Dans ce cas, l'utilisateur est tenté de désactiver la solution si celle-ci est trop invasive. Il est aussi possible d'attaquer la fonction de score malicieuse. Un processus peut s'arranger pour rester en dessous du seuil critique : ajout d'entropie faible, ne chiffre qu'une petite partie d'un fichier, etc. Seulement, s'il n'utilise qu'une de ces techniques, il sera malgré tout détecté grâce à la combinaison des autres indicateurs (e.g., énumération de nombreux répertoires). Pour ne pas être détecté, un logiciel de rançon doit combiner des techniques anti-REDEMPTION. Et même en combinant ces techniques, le logiciel de rançon doit parcourir le système de fichiers et accéder en écriture aux fichiers utilisateurs. Sa seule chance est de chiffrer peu de fichiers ou d'introduire des pauses entre chaque phase malicieuse. C'est à ce moment qu'intervient le module de restauration, mais sa configuration semble être fonction de l'utilisateur. Il faut attendre que les fichiers chiffrés mis en cache remplacent les originaux présent sur le disque après une période d'inaction de X heures ou minutes du logiciel malveillant. Ce qui limite grandement les chances de succès de la prise d'otage. Un antivirus traditionnel a ainsi plus de temps pour le détecter mais aussi un utilisateur. Nous pouvons également imaginer qu'un logiciel de rançon lance une myriade de processus (e.g. la nuit) dont le but est de ne chiffrer qu'un seul fichier utilisateur puis meurt. Une attaque par déni de service (i.e., sature l'espace de stockage) sur la zone protégée est également possible. Les auteurs ne sont pas dupes : il est toujours la possibilité de passer à travers un mécanisme de défense, c'est la course à l'armement.

REDEMPTION est la première contre-mesure temps réel à réellement adresser le problème des performances en fournissant des tests plus aboutis que [103, 71]. Les auteurs ont réussi à faire coexister une **surveillance lourde** de toutes les IRPs, l'ajout d'un module utilisateur et de **bonnes performances**. Cela peut sembler antinomique au premier abord, mais les auteurs l'ont réalisé. Nous regrettons qu'ils n'aient pas utilisé une approche moins globale (i.e., IRPs), et que le binaire de ne soit pas fourni (i.e., idem pour [103, 71]), ne serait-ce que pour comparer les performances. Le taux de détection est de 100% avec un taux de faux positifs entre 0.5% et 0.8% en fonction des expérimentations. De plus, grâce à son mécanisme de cache sur les fichiers utilisateurs, aucun n'est perdu lors d'une attaque. L'expérience utilisateur est une grande première, très intéressante, qui manque de participants et qui aurait mérité un protocole plus étoffé.

Data Aware Defense (DaD) : Towards a Generic and Practical Ransomware Countermeasure [PDB+17]

En 2017, nous avons proposé dans Palisse *et al.* [PDB+17] indépendamment de Kharraz *et al.* [88] une contre-mesure temps réel appelée DATA AWARE DEFENSE qui adresse le problème des performances tout en conservant un excellent taux de détection. Notre contribution diffère de part sa conception des contres-mesures précédente. Les détails concernant son implémentation et les résultats sont discutés dans les chapitres 5 et 6 consacrés aux contributions de cette thèse.

Cette sous-section présente des contres-mesures temps réel et deux plateformes d'analyse automatique de logiciels de rançon. L'ensemble de ces solutions est basée sur la surveillance des I/Os du système de fichiers à partir desquelles sont calculées des heuristiques. Le **taux de détection est de 100**% pour les solutions temps réel. Trois séquences d'accès caractéristiques des logiciels de rançon ont été identifiées. Seulement ces motifs peuvent déclencher des **faux positifs** avec les applications qui ont un comportement proche des logiciels de rançon. Les auteurs utilisent donc une **combinaison d'indicateurs** pour écarter les applications bénignes. Malheureusement, c'est au **détriment des performances**. On peut regretter qu'aucune des solutions ne soit disponible. Celles-ci implémentent toutes un pilote de type "file system minifilter driver". Aucune ne surveille la MFT du système de fichiers NTFS comme suggéré par Kharraz *et al.* [89]. Le défi technique étant de taille. Il faut attendre Kharraz *et al.* [88] et Palisse *et al.* [PDB+17] pour bénéficier de contre-mesures **utilisables** par un utilisateur (i.e., surcoût acceptable).

2.3 Conclusion

Ce chapitre met en perspective les contributions sur les logiciels de rançon de 1996 à 2018. Nous partons d'une menace hypothétique en 1996 avec Young et Yung [112]. Celle-ci est bien réelle deux décennies plus tard. Les solutions basées sur la CryptoAPI ne sont pas retenues, car non-générique comme discuté sous-section 2.2.1. En 2015, Kharraz *et al.* [89] présentent deux solutions pour observer l'effet du code malveillant sur le système de fichiers. Cette approche a pour avantage d'être générique et agnostique quant à la nature des primitives cryptographique utilisées. Les contre-mesures utilisent ensuite des indicateurs de compromission pour détecter un comportement malveillant. Force est de constater que toutes se servent de nombreux indicateurs qui consomment des ressources : processeur, disque et mémoire. L'objectif étant de fournir : (1) un taux de détection optimal et (2) un taux de faux positifs très faible. Les auteurs oublient ainsi que (3) les performances de leur contre-mesure sont primordiales.

Le TABLEAU 2.2 fournit une liste des contre-mesures temps réels. On voit bien que celles-ci n'apparaissent que tardivement, c'est-à-dire en 2016. Cette même année, on compte au moins 6 publications sur les logiciels de rançon.

Les TABLEAUX 2.3 et 2.4 comparent les 4 contre-mesures temps réel basées sur les entrées-sorties du système de fichiers. Trois d'entre elles utilisent au moins 5 indicateurs de compromission tout en surveillant également au moins 5 IRPs. Notre contribution, Palisse *et al.* [PDB+17] se démarque en ne surveillant qu'une seule IRP et en n'utilisant qu'un seul indicateur de compro-

Tableau 2.2 – Liste des contre-mesures temps réels aux logiciels de rançon.

Stratégie	**Contribution**	**Nom usuel**	**Année**
Appels à la CryptoAPI	**Palisse *et al.* [PBL+16]**	-	2016
	Kolodenker *et al.* [93]	PayBreak	2017
	Genç *et al.* [81]	USHALLNOTPASS	2018
Entrées - sorties du système de fichiers	Scaife *et al.* [103]	CryptoDrop	2016
	Continella *et al.* [71]	ShieldFS	2016
	Kharraz *et al.* [88]	Redemption	2017
	Palisse *et al.* [PDB+17]	Data Aware Defense	2017

mission. Il est difficile de comparer les résultats, en effet les corpus ne sont pas disponibles.

Il est également difficile de comparer les performances des contre-mesures, leur protocole expérimental est incomplet ou non reproductible. Par exemple, [103, 71] n'utilisent pas un seul outil standard pour l'évaluer les performances. De plus, [103, 71, 88] ne sont pas disponibles, contrairement à [PDB+17] [16]. Deux contre-mesures disposent d'une évaluation des performances reproductible : [88, PDB+17]. Celles-ci sont également les seules à être utilisable par un utilisateur dans des cas d'usage réels. On peut regretter que [88] ne précisent pas s'ils utilisent un disque dur ou un SSD. L'utilisation d'un disque dur est moins pénalisant car celui-ci est plus lent, par conséquent l'ajout d'un temps de latence peut-être imperceptible. Un SSD étant plus performant, celui-ci sera plus sensible à l'introduction d'une latence même faible. Les contributions [103, 88] emploient une plateforme d'analyse automatique, *Cuckoo Sandbox*, pour automatiser l'analyse des échantillons. [71] utilise des machines virtuelles sans donner davantage de précisions. Nous sommes les seuls dans Palisse *et al.* [PDB+17] à proposer un environnement de test constitué de machines nues (i.e., "bare-metal"). Dans le chapitre suivant, je vais présenter notre plateforme d'analyse automatique de logiciels malveillants : Malware - O - Matic.

16. http://people.rennes.inria.fr/Aurelien.Palisse/DaD.html

Tableau 2.3 – Comparaison (1) des indicateurs de compromission, des IRPs et des résultats, des contre-mesures temps réels aux logiciels de rançon basés sur les entrées-sorties du système de fichiers.

Contribution	**Indicateur**	**IRP**	**Détection**	**Résultat**
Scaife *et al.* [103] CryptoDrop	entropie de Shannon "file type tunneling" changement de type suppression "fuzzy hash"	READ WRITE INFORMATION CREATE CLOSE au moins 5 IRPs	combinaison des indicateurs processus	TP 100% FP *incomplet* 492 échantillons
Continella *et al.* [71] ShieldFS	entropie de Shannon "file type tunneling" # de répertoires énumérés* # de fichiers lus* # de fichiers écrits* # de fichiers renommés ou déplacés* *subi une normalisation	READ WRITE INFORMATION CREATE CLOSE au moins 5 IRPs	apprentissage automatique supervisé processus	TP 100% FP 0.038% 688 échantillons
Kharraz *et al.* [88] Redemption	entropie de Shannon changement de type suppression réécriture énumération des répertoires et fichiers fréquence d'accès aux données utilisateur	*XXX* toutes les IRPs	fonction de score malicieuse processus	TP 100% FP 0.5% et 0.8% 1174 échantillons
Palisse *et al.* [PDB+17] Data Aware Defense	test du Khi-deux	WRITE **une seule IRP**	test d'hypothèse nulle **fil d'exécution**	TP 99.37% FP *incomplet* 798 échantillons

Tableau 2.4 – Comparaison (2) du protocole expérimental et des performances, des contre-mesures temps réels aux logiciels de rançon basés sur les entrées-sorties du système de fichiers.

Contribution	**Environnement de test**	**Corpus**	**Performance**
Scaife *et al.* [103] CryptoDrop	Cuckoo Sandbox machine virtuelle	492 échantillons 15 familles	*"with future optimizations, CryptoDrop can be run on a live system"* latence moyenne entre 1 et 16 ms les tests sont non reproductibles
Continella *et al.* [71] ShieldFS	- machine virtuelle	688 échantillons 12 familles	*"... we barely perceived it (i.e., overhead) while using a machine protected by ShieldFS"* surcoût entre 26% et 380% utilisation d'un disque dur les tests sont non reproductibles
Kharraz *et al.* [88] Redemption	Cuckoo Sandbox machine virtuelle	1174 échantillons 29 familles	outil standard : IOzone surcoût entre 3.4% et 8.7% disque dur ou SSD ? les tests sont reproductibles **utilisable**
Palisse *et al.* [PDB+17] Data Aware Defense	Malware - O - Matic **machine nu**	798 échantillons 30 familles	outil standard : CrystalDiskMark, PCMark et Windows Performance Toolkit surcoût entre 1.8% et 7.1% latence moyenne 11.7 μs utilisation d'un SSD les tests sont reproductibles **utilisable**

Chapitre 3

Malware - O - Matic : un bac à sable “bare-metal”

Dans le chapitre précédant, les contributions automatisent l'analyse des logiciels de rançon en s'appuyant sur des machines virtuelles, principalement par simplicité mais aussi pour permettre de tester des milliers d'échantillons quotidiennement. Or, un grand nombre des échantillons testés ne s'exécutent pas. Une des raisons invoquée par les auteurs est la détection par le logiciel malveillant qu'il se trouve dans un environnement virtualisé. Ce chapitre présente une plateforme d'analyse automatique de logiciels malveillants : Malware - O - Matic. Celle-ci est basée sur des machines nues. Nous proposons une plateforme transparente et souhaitons ainsi déclencher un comportement malveillant.

3.1 MoM : une plateforme d'analyse dynamique

L'utilisation d'un environnement contrôlé est nécessaire quand on souhaite analyser un grand nombre d'échantillons. Nous avons fait le constat que les solutions actuelles (e.g., *Cuckoo Sandbox*) introduisaient de nombreux biais lors des analyses. La solution que nous proposons est donc constituée uniquement de machines nues, tout en conservant les capacités d'une plateforme traditionnelle. Celle-ci est construite autour de trois outils open source : *Viper* [110], *Clonezilla* [69] et *Scrapy* [36].

MoM est composée d'un serveur maître Linux et de 5 machines de test équipées de SSD. Bien entendu, cette plateforme dispose d'une autorisation particulière du réseau Renater. MoM est accessible par accès physique uniquement dans le Laboratoire de Haute Sécurité de Rennes à Inria. Le serveur maître offre les fonctionnalités suivantes :

— récupération automatique des échantillons depuis des dépôts en ligne,

— orchestration des campagnes de tests,

— restauration des machines nues,

— sauvegarde des données collectées.

Une fois une campagne de test lancée, celle-ci ne nécessite plus l'intervention d'un opérateur. Les machines nues fonctionnent sans interruption. Après chaque analyse, une image complète du système est restaurée puis de nouveau infectée, etc. MoM dispose d'un réseau dédié à Inria, celui-ci est connecté à Internet sans filtrage particulier. Les machines sont équipées d'adresses privées de classe B. Un pare-feu joue le rôle de NAT. La machine maître n'est pas connectée à Internet [1] pour des raisons de sécurité.

```
deb@lhs-cl:~/viper-1.2$ ./v

       (_)
  _   _ ____  _____  ____
 | | | |  _ \|  __ \|  __)
  \ V /| | | | |__| | |
   \_/ |_| |_|_____/|_| v1.2
       |_|

You have 38136 files in your default repository
viper > find tag works -l 10
+----+----------------------------------+-----------------------+
| #  | MD5                              | Tags                  |
+----+----------------------------------+-----------------------+
| 1  | c6c57442bc117d2a482e92ecc9ff44cd | works, cerber         |
| 2  | 3e946556104bc76dbd0bde0e755780f1 | works, kqx.aqirde     |
| 3  | 7bbb346484186447fb1d085e6942b56b | works, razy.64168     |
| 4  | 9de672c9f2833ddf7b81aee25d1d9329 | works, cerber         |
| 5  | 53e4f25062114b24e7d8c3379a786703 | works, agent.btpr     |
| 6  | aed7120328d1178bb11ae8a029fc6854 | works, generic.17216321 |
| 7  | 73c9118fc4f8762e8b0f831dab7b576f | works, razy.64168     |
| 8  | d48c9f6460afccb8024c9ff4960e36ff | works, cerber, nsis   |
| 9  | 58dd66e7bf07aa918d446cf7fa6f008a | works, cerber         |
| 10 | 8bc5a56eba78e08ce01f878d0d5f87f9 | works, razy.76775     |
+----+----------------------------------+-----------------------+
viper > tags
+---------------------------+-----------+
| Tag                       | # Entries |
+---------------------------+-----------+
| untested                  | 35690     |
| broken                    | 2289      |
| x64_win7                  | 152       |
| x64_win10                 | 108       |
| works                     | 73        |
| locky                     | 39        |
| checking_3                | 39        |
| checking_2                | 32        |
| xorist                    | 9         |
| cerber                    | 6         |
| teslacrypt                | 5         |
```

FIGURE 3.1 – Interface de Viper.

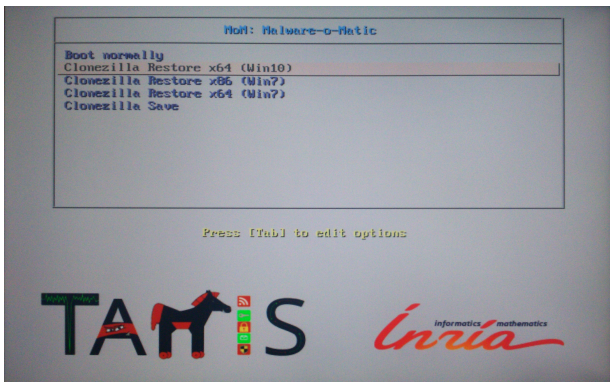

FIGURE 3.2 – Écran de démarrage de MoM.

Chaque analyse est précédée par l'exécution au démarrage de la machine victime d'un script téléchargé depuis le serveur maître. Ce script contient les instructions de la machine victime pour l'analyse (e.g., temps). La durée de chaque analyse est modulable et une activité minimale est simulée : déplacement de la souris, frappe du clavier et ouverture du navigateur. De plus, afin de faciliter l'exhibition d'un comportement malveillant, le contrôle de compte d'utilisateur (UAC) est désactivé, permettant ainsi à une application d'obtenir les droits administrateurs. MoM dispose de deux modes de fonctionnement : passif ou actif.

3.1.1 mode passif

Le premier mode a pour objectif de déterminer l'état de l'échantillon, c'est-à-dire si celui-ci est encore "vivant". Pour cela, les images système disposent d'un environnement réaliste, composés de fichiers récupérés depuis *Digital Corpora* [72]. Sur chaque image, une vingtaine d'extensions différentes sont présentes, mais aussi plus de 10k fichiers utilisateurs. Des applications tierces sont installées et le système est mis à jour régulièrement. Une fonction de hachage vérifie l'intégrité des fichiers de l'utilisateur après chaque analyse. Si celle-ci ne correspond pas à la valeur de référence, alors l'échantillon est considéré comme actif. MoM dispose de trois images différentes : Windows 7 x86, x64 et Windows 10 x64.

1. une exception : quand on récupère des échantillons en ligne

3.1.2 mode actif

Le second mode n'exécute que des échantillons actifs, son objectif est d'observer le comportement des échantillons sur le système de fichiers. À la fin de l'analyse, une trace contenant les I/Os est récupérée par le serveur maître. La capture des I/Os est obtenue grâce à un pilote de type "file system minifilter driver" développé par mes soins. Celui-ci transmet à une application dans l'espace utilisateur les I/Os interceptées. Cette application incorporée dans le script de démarrage, conserve en mémoire vive les informations reçues. L'objectif est de ne pas se faire chiffrer les traces en les sauvegardant sur le disque. À la fin de l'analyse une trace au format JavaScript Object Notation (JSON) compressée est envoyée au serveur maître. Compte tenu de la taille finie de la mémoire vive, il est nécessaire de limiter l'analyse à environ 10 ou 20 minutes. Cela dépend de l'activité de l'échantillon et du nombre d'I/O request packets différentes interceptées (e.g., écriture, lecture, etc). Sinon, le manque d'espace entraîne la perte de la trace. La trace au format JSON est ensuite facilement exploitable post-mortem pour comprendre le fonctionnement du logiciel de rançon. Les informations y sont stockées avec la granularité d'un fil d'exécution (i.e., thread). Une trace non compressée fait de 20 Mo à 200 Mo. Chaque trace est constituée de texte, celle-ci se compresse efficacement. Les traces sont également analysées manuellement, pour éliminer les échantillons au comportement similaire ou simplement qui ne ce sont pas déclenchés une seconde fois. Le script de démarrage est responsable de la récupération du pilote sur le serveur maître puis de son lancement.

Le script de démarrage est téléchargé par chaque machine "victime" grâce à une tâche planifiée. Celui-ci est un script Python s'exécutant en arrière plan. L'utilisation de *Clonezilla* a deux objectifs : (1) créer des images saines et (2) restaurer les images après infection. Il est possible de restaurer les partitions : GUID partition table ou master boot record. Bien pratique, quand on analyse Petya par exemple. Compte tenu de la configuration actuelle de MoM il faut 10 minutes pour restaurer un seul système. La restauration en parallèle de plusieurs machines entraîne une baisse des performances. À noter que chaque machine nue est indépendante, la restauration n'est donc pas collective. Il s'agit d'un choix de conception, permettant de lancer plusieurs campagnes de test différentes de manières concurrentes. En paramétrant la durée d'une analyse à 20 minutes, il est possible d'analyser 360 échantillons par jour.

Les échantillons sont récupérés depuis les trois sites suivant : *VirusShare* [42], *Malekal* [25] et *Malwr* [26]. Nous récupérons en moyenne 10 nouveaux échantillons par jour sur *VirusShare* [2]. La plateforme est donc capable d'analyser les échantillons dont nous pouvons disposer. Les compagnies d'antivirus disposent de davantage d'échantillons, mais aussi de plus grand moyens, ce qui leur permet d'adapter leurs outils d'analyse. Un outil d'extraction de contenu, *Scrapy*, récupère les échantillons automatiquement à partir de mot-clés. *Viper* conserve pour chaque campagne l'état des échantillons, c'est notre base de données. Celle-ci est stockée sur un NAS. Pour chaque échantillon des "tags" lui sont attachés. On garde une trace de l'image sur laquelle l'échantillon s'est exécuté, mais aussi de son statut. Un module supplémentaire qui soumet les hachés à *VirusTotal* détermine la famille des échantillons.

2. un téléchargement quotidien sur *VirusShare* pendant une semaine avec le mot-clé "ransomware"

L'utilisation de trois outils open source rend notre configuration reproductible. MoM est principalement le fait de configurations réseaux et systèmes. Des sauvegardes de notre serveur maître et du NAS sont réalisées périodiquement. En effet, il est plausible qu'un logiciel de rançon tente de se propager sur le réseau local de la plateforme. MoM cible de part ses deux modes de fonctionnement les logiciels de rançon mais rien ne s'oppose à l'ajout de scénarios d'analyse supplémentaires. L'analyse de certains échantillons déclenche un redémarrage intempestif ou "freeze" le système. Le redémarrage intempestif n'est pas un problème. Par contre, si le système "freeze" une machine est consommée sans que MoM ne puisse rien y faire. Pour résoudre cela, nous avons récemment fait l'acquisition de matériel permettant de commander l'alimentation des machines victimes.

Concernant les limitations, le contenu des images disque est statique. Par exemple, le nom de la machine et de l'utilisateur sont identiques pour toutes les analyses de déroulant sur la même image. La date et l'heure sont par contre mises à jour. Le fait d'utiliser des machines nues complique une mise à l'échelle de la plateforme plus importante. Une autre limitation, les pilotes doivent être signés sur les versions 64-bit de Windows. Or, nous n'avons pas de certificat de code. Le mode actif est donc privilégié sur les machines 32-bit qui ne nécessitent pas une signature valide. Il est possible d'utiliser un pilote 64-bit signé par un certificat de test de *Visual Studio*, il faut pour cela modifier des paramètres Windows, ce qui peut alerter un logiciel malveillant.

3.2 Conclusion

Malware - O - Matic est une plateforme d'analyse de logiciels malveillants constituée de machines nues. Des solutions similaires ont déjà été proposées dans la littérature. On peut citer BareBox [92] et Nvmtrace [31]. L'utilisation de l'émulation, de la virtualisation ou d'un hyperviseur facilite l'évasion des logiciels de rançon. En effet, celles-ci ne sont pas transparentes. Dinaburg *et al.* [73] utilisent ce terme pour désigner un système d'analyse qui est indistinguable d'un système normal. Ils identifient 5 exigences :

1. le système d'analyse a des privilèges supérieurs à l'échantillon,
2. aucun effet de bord n'est perceptible pour l'échantillon à son niveau de privilège,
3. la sémantique des instructions est identique,
4. une gestion des exceptions transparente,
5. une mesure du temps identique,

MoM respecte 4 de ces 5 exigences. La seconde n'est pas tenue, un logiciel de rançon peut lister les pilotes qui s'exécutent sur le système s'il obtient les droits administrateur est ainsi détecter le pilote de supervision. Je pourrais utiliser une technique de "rootkit" pour cacher le pilote.

D'autres facteurs sont à prendre en compte pour éviter que les logiciels malveillants n'altèrent volontairement leur comportement. On peut noter, les artefacts environnementaux et d'analyses, mais aussi les tests de Turing inversés. Rossow *et al.* [102] parlent de "réalisme" et Miramirkhani *et al.* [99] présentent une liste d'artefacts qualifiés de "wear and tear". Il s'agit d'artefacts crées

directement ou indirectement par l'utilisation passée d'une machine par un utilisateur. Miramirkhani *et al.* [99] détectent ainsi avec une précision de 92.86% un environnement artificiel par rapport à une vraie machine utilisateur. Les artefacts qu'ils ont identifié sont très nombreux, il est donc difficile de tous les simuler. Ils proposent deux méthodes pour créer un environnement réaliste : (1) utiliser une image système d'un vrai utilisateur mais cela pose un problème de respect de la vie privée et (2) simuler automatiquement le comportement d'un utilisateur. Un logiciel malveillant peut également déterminer passivement si un utilisateur est présent ou lui présenter un message. Kharraz *et al.* [87] s'intéressent au réalisme, ils génèrent automatiquement des images saines dont le contenu est créé à la volée. Mais cela ne concerne que le système de fichiers, leur solution étant dédiée aux logiciels de rançon.

Une plateforme de machines nue transparente n'est donc pas suffisante. On conserve l'avantage de ne pas être sensible à : (1) "timing attack", (2) artefacts CPU et (3) VMM détection. Bulazel *et al.* [68] considèrent que la création d'un environnement réaliste est le plus dur à réaliser. MoM ne déploie pas de technique pour mitiger l'évasion. Par exemple, l'utilisation de "stalling code" est un problème, la durée d'une analyse sur machine nue est limité en raison de son coût. L'introspection d'une machine nue est plus complexe, car aucun mécanisme de virtualisation ou d'émulation ne permet de contrôler les instructions exécutées, nous avons opté pour un pilote.

Rossow *et al.* [102] présentent un ensemble de bonnes pratiques pour analyser les logiciels malveillants. Nous avons essayé de les respecter. Notamment, une présentation claire de la plateforme d'analyse, MoM, pour que les expérimentations soient reproductibles. Les considérations légales, éthiques et de confinement sont également de mise : demande d'utilisation particulière du réseau Renater et utilisation d'un réseau dédiée à Inria.

Symantec [12] déclare que de plus en plus de logiciels malveillants n'hésitent pas à se déclencher dans une machine virtuelle, du fait de leur utilisation dans le monde professionnel par exemple. Les TABLEAUX 3.2 et 3.1 semblent le confirmer. En effet, le pourcentage de perte, soit le nombre d'échantillons inactifs parmi les échantillons récupérés initialement, est plus élevé en moyenne avec MoM que la moyenne des contributions du domaine. Ce pourcentage de perte est de 83.74% en moyenne avec MoM et de 82.67% en moyenne pour les contributions du domaine. On peut noter, que nous avons utilisé une archive provenant de *VirusShare* spécialisée dans les logiciels de rançon. Celle-ci contenait de nombreux échantillons inactifs ou des logiciels malveillants au comportement similaire qu'il a fallu éliminer. Certains logiciels de rançon communiquent avec des serveurs distants avant de lancer une attaque. Ceux-ci ont une durée de vie courte. Les échantillons sont donc actifs pour une fenêtre temporelle à la durée variable, c'est-à-dire le temps d'une campagne d'attaque. Continella *et al.* [71] sont les seuls à ne pas donner le nombre d'échantillons récupérés initialement ainsi que leurs provenances. Il est difficile de comparer les performances de MoM avec les plateformes utilisées par les contributions du domaine. Les corpus ne sont pas disponibles. Les sources privilégiées TABLEAU 3.2 sont *VirusTotal* et *VirusShare*. Le TABLEAU 3.1 montre que nous n'avons pas utilisé *VirusTotal*. En fonction des sources, des biais sont introduits.

Le fait que MoM produise des stimuli simples et répétitifs et que l'environnement de l'utilisateur contiennent des artefacts, déclenche sans doute des réflexes d'évasion chez certains échantillons. Afin d'améliorer la plateforme, les points ci-dessus devront être travaillés. L'annexe A

contient la liste des échantillons actifs de notre corpus. Les hachés sont disponibles sur mon site académique[3]. *AVclass* [104] est utilisé pour identifier les familles. L'objectif de cet outil est de fournir automatiquement le label le plus vraisemblance à un échantillon. *AVclass* est capable d'identifier les alias utilisés par les antivirus pour un même échantillon, et normalise ensuite les labels, réglant ainsi les problèmes de labels inconsistants.

Tableau 3.1 – Pourcentage de perte que nous avons expérimenté lors de la sélection des échantillons : logiciel de rançon actif. MoM est utilisé pour exhiber le comportement malveillant.

Campagne	**# d'échantillons récupérés**	**# d'échantillons actifs**	**Pourcentage de perte**
une archive *VirusShare* dédiée aux logiciels de rançon le 16/02/2017	18640	627	96.6%
récupération automatique src : *VirusShare*, *Malekal* et *Malwr* le 11/04/2017	860	171	80.11%
récupération automatique src : *VirusShare*, *Malekal* et *Malwr* le 02/06/2017	970	247	74.53%
moyenne			**83.74**%

3. http://people.rennes.inria.fr/Aurelien.Palisse/corpus.html

Tableau 3.2 – Comparaison du pourcentage de perte des contributions du domaine lors de la sélection des échantillons : logiciel de rançon actif. Celles-ci utilisent toutes des machines virtuelles.

Contribution	**# d'échantillons récupérés**	**# d'échantillons actifs**	**Pourcentage de perte**
Scaife *et al.* [103] CryptoDrop	2663	492	81.5%
Continella *et al.* [71] ShieldFS	-	688	-
Kharraz *et al.* [88] Redemption	9432	1174	87.5%
Kolodenker *et al.* [93] PayBreak	713	107	84.9%
Genç *et al.* [81] USHALLNOTPASS	2263	524	76.8%
moyenne			**82.67**%

Chapitre 4

CryptoAPI et logiciels de rançon : détection et limitation

Ce chapitre propose la première contre-mesure aux logiciels de rançon basée sur la CryptoAPI. Cette contribution : *Ransomware and the Legacy Crypto API* [PBL+16] a été publiée en 2016 à la conférence *CRiSIS*. Seulement quatre publications concernent les logiciels de rançon à cette date. Aucune n'implémente de contre-mesure [1]. Young et Yung [112] et Kharraz *et al.* [89] proposent de surveiller les accès aux primitives cryptographiques. La CryptoAPI de Microsoft est alors un choix privilégié pour les logiciels de rançon qui chiffrent les documents utilisateur. Principalement pour les trois raisons suivantes : (1) facilité d'utilisation, (2) robustesse des primitives et (3) portabilité. Notre objectif dans ce chapitre est donc d'implémenter une contre-mesure efficace qui s'intègre dans la CryptoAPI de manière transparente.

4.1 Intégration d'une contre-mesure dans la CryptoAPI

On souhaite prévenir un usage malveillant de la CryptoAPI et ainsi empêcher que les fichiers utilisateur soient altérés. Pour cela, le principe général est l'interception et le stockage d'information pour un usage ultérieur.

4.1.1 Présentation de la CryptoAPI

À partir de Windows 95, les systèmes d'exploitation Windows disposent d'une API qui fournit des services cryptographiques. Ces services sont disponibles pour les applications de l'espace utilisateur à travers la CryptoAPI. Les primitives cryptographiques sont embarquées dans des librairies dynamiques qui représentent en fait des : "Cryptographic Service Providers" (CSPs). Chacun de ces fournisseurs de services est spécialisé dans une classe d'algorithme (e.g., signature). La CryptoAPI offre une couche d'abstraction sur les primitives et permet ainsi leurs utilisations par des non-spécialistes. Il est possible d'ajouter des fournisseurs de services cryptographiques. Une entreprise peut par exemple, implémenter ses propres algorithmes dans un module matériel

1. Nicoló *et al.* [63] propose HelDroid sur *Android*

de sécurité (HSM). Un fournisseur de services doit être signé par une autorité de confiance pour être pleinement opérationnel.

Il est important de noter qu'il n'est pas nécessaire de s'authentifier pour accèder à l'API. L'authentification repose exclusivement sur le système d'exploitation. Les clés générées par la CryptoAPI ne sont pas persistantes. Il faut utiliser une API distincte pour les stocker (i.e., Data Protection API). À chaque fois que le système instancie une librairie dynamique, le code est partagé mais pas les données. Les données sont uniquement accessibles au processus correspondant, ce qui permet de ne pas exposer le matériel sensible. Pour davantage de détails se référer à [7].

La CryptoAPI est encore présente sur les systèmes modernes pour des raisons de rétrocompatibilités. Elle est qualifée de "deprecated". Il est recommandé d'utiliser sa remplaçante apparue avec Windows Vista : "Cryptography API: Next Generation" (CNG). Par exemple, *OpenSSL* emploie préférentiellement CNG pour initialiser un générateur de nombres pseudo-aléatoires [2]. La CryptoAPI ne dispose pas d'algorithmes de chiffrement à base de courbes elliptiques (ECC). De plus, CNG implémente la suite d'algorithmes B. Il s'agit d'un standard Américain du National Institute of Standards and Technology (NIST) pour protéger les informations. Il est aussi possible d'utiliser CNG dans le noyau, les mêmes opérations sont supportées.

4.1.2 Protection du système

La question principale est comment se prémunir contre une utilisation malicieuse de la CryptoAPI sans employer de méthode invasive ? Pour cela, notre contre-mesure se doit d'être la plus simple et transparente possible. Les logiciels de rançon qui évoluent dans l'espace utilisateur tout comme notre contre-mesure ne doivent pas se douter de sa présence. Il est possible d'utiliser la librairie *Detours* [**?**] de Microsoft pour placer des crochets (i.e., hook). Cette méthode déjà largement employée est intrusive. Il est nécessaire d'accèder à la mémoire du processus suspect et de modifier des instructions. Nous choisissons donc d'implémenter notre propre fournisseur de services cryptographiques et de l'incorporer au système. On dispose ainsi d'une position privilégiée et légitime pour observer l'usage de la CryptoAPI. De plus, l'utilisation de notre CSP est non-intrusive et transparente pour une application utilisateur. Il est tout de même nécessaire de fournir de "vrai" services cryptographiques. En effet, des applications légitimes vont utiliser nos services. La librairie *OpenSSL* implémente les primitives cryptographiques de notre CSP. Le développement d'un CSP nécessite l'utilisation d'un kit de développement particulier : Cryptographic Provider Development Kit [8]. *OpenSSL* 1.0.2e de décembre 2015 est utilisé. Une édition de liens statique est réalisé avec notre CSP. Notre CSP fournit les services suivants :

— AES avec les modes de chaînage CBC et ECB,

— RSA PKCS#1 v1.5 et v2.0,

— RSA textbook,

— MD5, SHA1 et SHA256,

— source d'aléa.

2. https://github.com/openssl/openssl/blob/master/crypto/rand/rand_win.c

La suite d'algorithmes ci-dessus est suffisante pour adresser les besoins des logiciels de rançon et des applications utilisateur. Notre CSP, est une librairie dynamique intégrée dans le système, 32 ou 64-bit, en le plaçant dans le répertoire *System32* et en modifiant des clés de registre [3]. Pour cela, les droits administrateurs sont nécessaires et *regsvr32.exe* est employé. Comme expliqué précédemment, le système ne peut pas avoir confiance et donc utiliser un fournisseur de services cryptographiques qui n'est pas signé. Pour les besoins de notre preuve de concept, nous avons patché le binaire responsable de l'authentification des CSPs. La FIGURE 4.1 illustre l'intégration de notre contre-mesure et son utilisation par une application malicieuse.

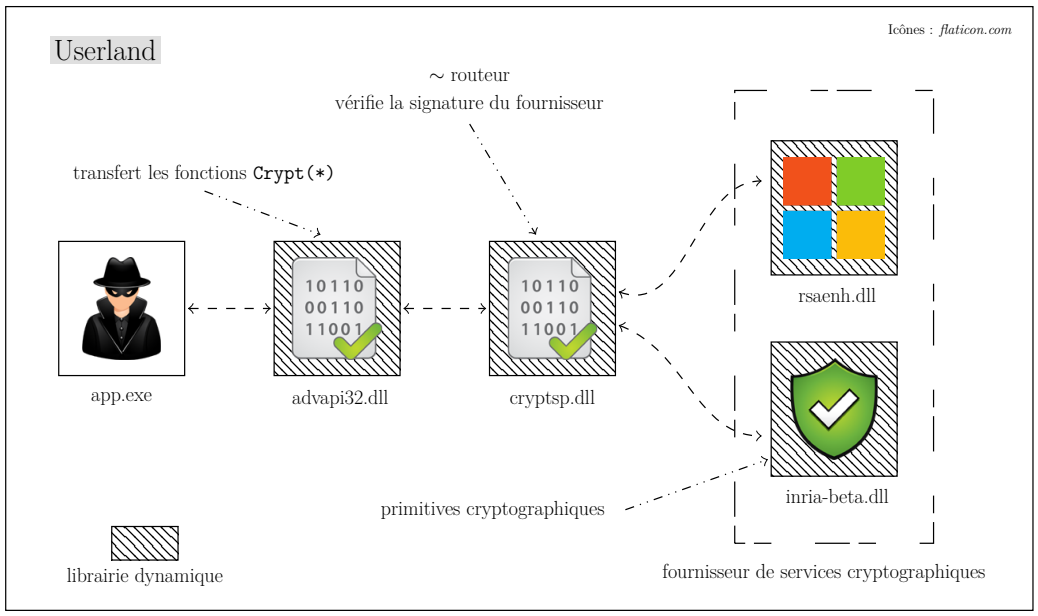

FIGURE 4.1 – Intégration de notre contre-mesure dans la CryptoAPI.

Une application utilisateur fait appel à la CryptoAPI via les fonctions *Crypt(*)* exportées par la librairie *advapi32.dll*. Il s'agit en fait d'un saut vers la librairie *cryptsp.dll*. Celle-ci est en charge de l'authentification des CSPs et du "routage" des appels vers le CSP correspondant. Initialement, un système ne possède que les CSPs de Microsoft. En ajoutant notre propre CSP, nous devons nous assurer de son utilisation. Nous forçons donc son emploi au détriment : (1) du désir de l'utilisateur et (2) du fournisseur par défaut. En effet, un logiciel de rançon peut choisir un CSP lors de l'appel à *CryptAcquireContext*, voir FIGURE 4.2. Cette fonction retourne une référence (i.e., handle) vers le CSP souhaité, ce qui permet ensuite de l'utiliser. Quand une application ne précise pas le CSP qu'elle souhaite utiliser, le CSP par défaut est employé. Celui-ci est déterminé en fonction d'une clé de registre. Pour rediriger tous les appels des fonctions *Crypt(*)* vers notre CSP, nous procédons comme suit : (1) patch de *cryptsp.dll* pour diriger les appels de *CryptAcquireContext* vers le CSP par défaut, et ce même si l'utilisateur fournit de manière explicite le nom du CSP et (2) paramètrage du registre pour placer notre CSP en position par défaut. Les patchs sur *cryptsp.dll* sont visibles FIGURE 4.3.

3. HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Defaults\Provider

```
BOOL WINAPI CryptAcquireContext(
  _Out_   HCRYPTPROV *phProv,
  _In_    LPCTSTR   pszContainer,
  _In_Opt LPCTSTR   pszProvider,
  _In_    DWORD     dwProvType,
  _In_    DWORD     dwFlags
  );
```

FIGURE 4.2 – Prototype de *CryptAcquireContext*.

Si l'argument *pszProvider* est fourni, celui-ci désigne de manière explicite le fournisseur. *dwProvType* définit le type du CSP. Nous avons implémenté un CSP de type : *PROV_RSA_AES*.

```
This difference file has been created by IDA

cryptsp.dll
00003944: 3B 90 // NOP
00003945: D3 90
00003946: 0F 90
00003947: 84 E9 // JMP vers CSP par défaut
00003CE1: E8 B8 // MOV EAX, 1
00003CE2: C2 01
00003CE3: F7 00
00003CE4: FF 00
00003CE5: FF 00
```

```
This difference file has been created by IDA

cryptsp.dll
0000000000002B23: 4C 90 // NOP
0000000000002B24: 3B 90
0000000000002B25: E3 90
0000000000002B26: 0F 90
0000000000002B27: 84 E9 // JMP vers CSP par défaut
000000000000329D: E8 B8 // MOV EAX, 1
000000000000329E: 26 01
000000000000329F: 44 00
```

FIGURE 4.3 – Patch de *cryptsp.dll* 32 et 64-bit.

Les patchs ci-dessus ont été réalisé avec IDA [18]. La librairie est modifiée à deux endroits. On réécrit l'appel à la fonction de vérification de la signature par : "*mov eax, 1*" soit "\xB8\x01". La seconde modification est la mise en place d'un *jump* relatif inconditionnel vers le branchement du CSP par défaut. Pour cela, on place plusieurs *nop* avec "\x90" puis *jump* c'est-à-dire "\xE9". Notre fournisseur de services cryptographiques est développé pour Windows 7. Les patchs de la FIGURE 4.3 conviennent donc à la librairie *cryptsp.dll* pour ce même système d'exploitation [4].

Le comportement malveillant des logiciels de rançon doit être stoppé et les dommages tempérés. C'est pour cela, que la restauration des fichiers chiffrés est nécessaire. Nous avons le contrôle sur la génération des clés, le chiffrement, la signature, le générateur de nombres aléatoires, etc. Lors du chargement de notre CSP, aucune clé n'est présente en mémoire. La suppression et/ou l'exportation du matériel sensible intervient une fois que les opérations malicieuses sont terminées. On surveille donc les appels à la CryptoAPI et si un comportement malveillant est détecté : (1) le matériel sensible encore en mémoire est stocké et (2) le processus est terminé. À noter que notre CSP à un contrôle total sur les clés, c'est lui qui alloue et désalloue leur mémoire. En parallèle, les opérations des processus utilisateur dans notre CSP sont enregistrées dans un fichier à l'en-tête et l'extension d'un exécutable au format : Portable Executable (PE). L'objectif est d'empêcher le chiffrement de cette trace. Cette même trace permet ensuite de revenir sur

4. des différences mineures peuvent être observées en fonction des versions

les modifications malicieuses des fichiers. Notre contre-mesure est donc une solution temps réel avec une capacité de remédiation post-infection. Il est difficile de caractériser un comportement anormal ou malveillant. En effet, des applications légitimes peuvent reproduire un comportement supposé malveillant. Nous n'avons pas ajouté de module de prise décision à notre CSP. Nous laissons cela pour des travaux futurs. La CryptoAPI est marquée comme "deprecated", il est donc peu probable qu'elle soit utilisée par beaucoup d'applications et qui plus est moderne. Dans tous les cas, une utilisation intensive de la CryptoAPI et plus particulièrement des fonctions de chiffrement est suspecte.

Le déploiement de notre contre-mesure a pour objectif d'évaluer si les logiciels de rançon utilisent la CryptoAPI. Ensuite, il s'agit d'analyser leurs usages de la CryptoAPI. Puis, discriminer les logiciels de rançon des applications légitimes pour concevoir un module de prise de décision.

4.2 Expérimentations et résultats

Avant de présenter le protocole expérimental et les résultats de nos expérimentations, les limitations de notre contre-mesure doivent être discutées. Celles-ci impactent les résultats de nos expérimentations. La librairie *cryptsp.dll* est protégée, aussi modifier son code la rend invalide. Son haché est stocké dans un catalogue de sécurité qui est lui-même signé. On ne peut donc pas manipuler ce catalogue facilement. Un exemple de manipulation de la signature d'un exécutable est fourni dans [33]. Le moyen le plus simple de passer outre cette limitation, est de lancer le logiciel de rançon dans le même répertoire que la librairie patchée. En effet, lors de la résolution d'une librairie dynamique, le répertoire à partir duquel l'application est lancée est le premier à être vérifié [13]. Pour les besoins de nos expérimentations, cela nous convient. Dans l'hypothèse d'un déploiement dans le monde réel, ce n'est pas satisfaisant. L'intégration de notre CSP aurait été plus aisée si nous l'avions fait signé. Seul bémol, les applications qui désignent explicitement le CSP de Microsoft, dans ce cas, le fait de placer notre CSP en choix par défaut dans le registre n'est pas suffisant.

Les expérimentations se déroulent sur des machines nues équipées de Windows 7 et l'exécution de chaque échantillon est réalisé manuellement. À cette époque, la plateforme MoM présentée au chapitre 3 n'est pas encore opérationnelle. Une restauration complète des machines est effectuée après chaque analyse grâce à *Clonezilla* [69]. Les machines sont reliées à Internet et aucun outil d'analyse de logiciels malveillants n'est déployé. On souhaite ainsi être le plus transparent possible, seule notre contre-mesure est incorporée dans le système. Les échantillons sont récupérés depuis des dépôts en ligne : *VirusShare* [42], *Malekal* [25] et *Malwr* [26]. Le fait de procéder manuellement au test de chaque échantillon limite la taille de notre corpus. Nous récupérons 54 échantillons appartenant à 16 familles. 27 échantillons sont actifs, soit un pourcentage de perte de 50%. Comparé aux résultats du chapitre précédant (e.g., 74%), le pourcentage de perte est bien inférieur. Les échantillons récupérés sont plus récents, car faible en nombre, et leur analyse manuel avec un opérateur stimule davantage les logiciels de rançon.

Le TABLEAU 4.1 présente les résulats de nos expérimentations. Les familles les plus répandues en 2016 sont présentes : Cerber, CryptoWall et TeslaCrypt. Malheureusement, certaines familles ne possèdent pas un seul échantillon actif.

Tableau 4.1 – Les résulats de nos expérimentations sur des machines nues avec notre contre-mesure. La quatrième colonne donne le nombre d'échantillons actifs pour une famille. La dernière colonne indique si les échantillons utilisent la CryptoAPI. Si tel est le cas : ✓et ■ sont présents et la contre-mesure est efficace. Pour certaines familles, aucun échantillon n'est actif ■. Lorsque des échantillons d'une même famille sont divisées en plusieurs lignes, c'est le signe que des différences significatives ont été observées.

Famille	1re apparition	# échantillons	# échantillons actifs	CryptoAPI
Gpcode	2004	4	4	✓
		1	1	×
CryptoLocker	2013	7	-	
		2	2	✓
DirtyDecrypt	2013	1	1	×
CryptoWall	2013	5	-	
CTB-Locker	2014	4	4	×
TorrentLocker	2014	3	-	
ZeroLocker	2014	2	2	×
CryptoFortress	2015	3	3	✓
TeslaCrypt	2015	2	2	×
		1	1	×
		1	-	
CrypVault	2015	2	-	
DMA-Locker	2016	1	-	
Locky	2016	5	-	
HydraCrypt	2016	1	1	✓*
		1	-	
Petya	2016	2	2	×
Cerber	2016	1	1	×
		2	-	
JigSaw	2016	3	3	×
# total échantillons efficacité contre-mesure		54	27 (50%)	10 (37%)

* utilisation d'une fonction de hachage.

CTB-Locker, TeslaCrypt et Petya ne communiquent pas avec un serveur distant avant de chiffrer les documents utilisateur. Leurs taux de perte sont donc très faibles. 37% des échantillons utilisent la CryptoAPI. Cela correspond à 10 échantillons répartis en 4 familles. Les familles CryptoWall et Locky utilisent la CryptoAPI d'après les rapports [24, 4]. Nous ne pouvons le vérifier, car aucun de leur échantillon n'est actif. Les résultats sont donc médiocres. Les échantillons de Gpcode et CryptoLocker utilisent simplement la CryptoAPI. Ils acquièrent une référence vers le CSP, puis génèrent une ou plusieurs clés symétriques pour finalement exporter les secrets en dehors du CSP. Ils choisissent ensuite le mode de chaînage et le vecteur d'initialisation. Par défaut, c'est le mode CBC qui est utilisé avec un vecteur nul. Les opérations de chiffrement sont ensuite réalisées via des appels à *CryptEncrypt()*.

Les résultats sont peu significatifs pour deux raisons : (1) taille du corpus et (2) intégration de notre contre-mesure. Il aurait été préférable d'obtenir une signature valide pour notre fournisseur de services cryptographiques afin de ne pas avoir à patcher la librairie *cryptsp.dll*. Cela biaise les résultats. 63% des échantillons actifs n'utilisent pas la CryptoAPI. L'implémentation ou l'intégration de notre CSP pose peut-être problème à certains échantillons et explique le faible taux de détection.

4.3 Conclusion

Notre contribution est la première à proposer une contre-mesure aux logiciels de rançon basée sur la CryptoAPI dans le monde académique. Celle-ci est conçue pour être exécutable en temps réel avec une restauration des fichiers post-infection. Malheureusement, l'intégration de notre contre-mesure lors de nos expérimentation souffre de limitations. Celles-ci, peuvent être comblées en partie si notre CSP est signé. Nous obtenons un taux de détection de 37%.

Dans un rapport sur les logiciels de rançon, l'entreprise *Bromium* [4] en 2014 utilise la librairie Detours [**?**] de Microsoft pour crocheter des fonctions de la CryptoAPI. Cette approche plus fonctionnelle souffre de limitations théoriques bien plus importantes. En effet, il faut accéder à l'espace mémoire du processus puis modifier son code. Cette méthode est intrusive et augmente les chances d'évasion des logiciels malveillants. Les contributions suivantes dans la littérature basée sur la CryptoAPI : [93, 81, 95] utilisent toutes des crochets. Notre approche est bien plus furtive. Les seuls artefacts de notre contre-mesure sont dans le registre.

Nous avions prédit en 2016 que la CryptoAPI serait de moins en moins utilisée par les logiciels de rançon. Force est de constater que nous avions tort. En 2018, Genç *et al.* [81] bloque 94% des logiciels de rançon de leur corpus en ne s'intéressant qu'à la fonction *CryptGenRandom*. Le fait que la moitié des échantillons soit inactifs lors des expérimentations et que notre CSP ne soit pas intégré "légitimement" dans le système biaise nos résultats.

Il est intéressant de noter que la librairie CNG, remplaçante de la CryptoAPI, n'est pas utilisée par les logiciels de rançon. À ma connaissance, aucun logiciel de rançon ne l'utilise. Les raisons sont multiples : (1) aucun apport substantiel pour un attaquant, les algorithmes utiles sont déjà présent dans la CryptoAPI, (2) nombreux exemples de code, (3) partage de code, etc.

Les contre-mesures basées sur la CryptoAPI souffrent de deux limitations. La première est qu'elles s'exécutent dans l'espace utilisateur tout comme les logiciels de rançon, avec le même niveau de privilège. Il est donc possible pour un logiciel de rançon de détecter puis de rendre inopérente une contre-mesure. La seconde est que de nombreuses librairies cryptographiques sûres et éprouvées sont disponibles de nos jours (e.g., liaison statique). Les logiciels de rançon peuvent donc évader les contre-mesures basées sur la CryptoAPI.

La contribution de Genç *et al.* [81] montre néanmmoins que logiciels de rançon utilisent largement le générateur de nombres aléatoires de la CryptoAPI. Seulement il n'est pas possible de sauvegarder toutes ses sorties. Il est facile pour un logiciel de rançon d'utiliser un générateur tiers présent dans une librairie. De plus, Genç *et al.* [81] ne présentent aucune solution réaliste pour filtrer les accès des processus au générateur.

Dans le chapitre suivant, nous proposons une contre-mesure générique et agnostique basée sur une analyse comportementale du système de fichiers. Des opérations spécifiques aux logiciels de rançon peuvent y être observées à l'aide d'un ou plusieurs indicateurs de compromission.

Chapitre 5

Détection comportementale de logiciels de rançon

Ce chapitre s'intéresse au comportement des logiciels de rançon sur le système de fichiers. En effet, lors d'une attaque des opérations caractéristiques y sont observées. Cette approche est générique car peu importe les primitives cryptographiques utilisées, c'est l'effet sur le système qui est surveillé. Ce chapitre est découpé en deux parties. La première partie va présenter comment on peut intégrer une contre-mesure dans le système pour observer puis bloquer des requêtes sur le système de fichiers. La seconde partie, présente deux indicateurs de compromission qui permettent de distinguer une attaque d'un comportement bénin.

5.1 Une contre-mesure générique : File System Minifilter Driver

Nous nous intéressons ici au système d'exploitation Windows. La solution proposée peut être transposée à d'autres systèmes d'exploitation.

5.1.1 Surveillance du système de fichiers

Windows, comme la plupart des systèmes moderne, découpe sa mémoire en différentes régions ou chacune possède un niveau de privilège correspondant. Le mode noyau qui correspond au "ring-0" dispose d'un haut niveau de privilège. Le noyau est responsable en autres choses de la gestion des opérations sur le disque. Les pilotes s'exécutent généralement dans le "ring-0". Les applications standards quant à elles s'exécutent dans l'espace utilisateur, c'est-à-dire le "ring-3". Elles disposent d'un niveau de privilège inférieur et ne peuvent réaliser directement des opérations sur le disque. Pour se faire, les applications doivent passer par l'appel système correspondant du noyau Windows. Le noyau va ensuite gérer les opérations privilégiées puis rendre la main à l'application. Cette séparation existe pour des raisons de sécurité ; en effet elle garantit l'intégrité des ressources du système vis-à-vis de l'espace utilisateur. De plus, une erreur dans le noyau peut engendrer un plantage de la machine. Le code doit donc être robuste. Sur les versions 64-bit de Windows, les pilotes doivent être signés par une autorité de confiance pour fonctionner.

Au meilleur de ma connaissance en 2018, les logiciels de rançon s'exécutent dans l'espace utilisateur. C'est pourquoi une contre-mesure qui se positionne dans le noyau grâce à un pilote ne peut être manipulée par du code malveillant. En effet, les utilisateurs ont une vision limitée du système, il est donc plus facile de se cacher dans le noyau.

Les interactions des utilisateurs sur les fichiers sont associées à des opérations dans le noyau. Elles sont principalement du type I/O request packet (IRP). Il s'agit d'une structure de données qui encapsule le contexte de l'opération et fournit un niveau d'abstraction aux pilotes. Le gestionnaire des I/O assure la communication entre les applications utilisateurs et les matériels disponible (e.g., souris, carte réseau). Son architecture est constituée de différentes couches qui forment une pile ou "stack". Cette pile va de la requête initiée par l'application utilisateur vers le pilote du système de fichiers (e.g., ntfs.sys) puis jusqu'au pilote du matériel. Différents pilotes sont présents dans cette pile à des altitudes distinctes en fonction des besoins. La FIGURE 5.1 illustre cette architecture en pile pour le gestionnaire des filtres.

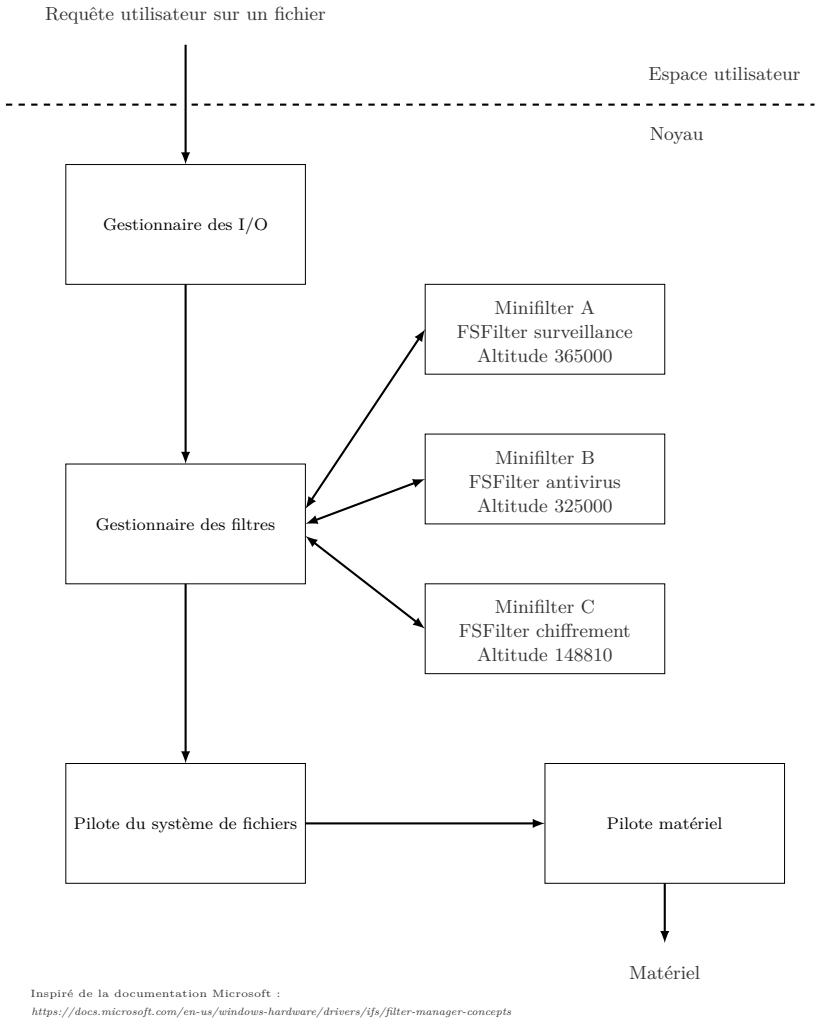

FIGURE 5.1 – Illustration simplifiée de l'architecture du gestionnaire des I/O et des filtres.

Microsoft offre la possibilité d'ajouter des pilotes tiers dans la pile du gestionnaire des I/O via un gestionnaire de filtres ou "filter manager". Nous pouvons donc nous intercaler de manière transparente entre une application qui réalise une opération sur le disque et le pilote matériel. Les artefacts ne sont détectables qu'avec les droits administrateurs et les effets de bords limités. Un kit de développement est disponible pour de tels pilotes qui sont de type : "File System Minifilter Drivers" [29]. Le concept de pile est également présent dans le gestionnaire de filtre. La position de chaque pilote "filtrant" est déterminée en fonction de son type. Par exemple, il existe des pilotes de chiffrement, de compression, etc [30]. Un pilote de chiffrement de disque est situé au plus prés du matériel pour ne pas interférer avec les autres pilotes. Cela permet par exemple, au pilote d'un antivirus situé plus haut dans la pile d'inspecter les requêtes en clair. Un pilote "filtrant" ou "minifilter" est capable d'inspecter toutes les opérations qui ciblent les disques. Dans ce contexte, il est possible de surveiller les requêtes d'écriture, de lecture, etc. Au-delà de l'inspection, nous pouvons modifier le contenu d'une requête, mais aussi la suspendre ou la bloquer. Cependant, il est nécessaire de filtrer de manière parcimonieuse les différents types de requête pour ne pas trop ralentir la machine. Les contre-mesures proposées par Continella *et al.* [71] et Scaife *et al.* [103] en sont des exemples. Elles ne peuvent être déployées en production.

5.1.2 Implémentation et architecture de notre contre-mesure

La détection d'un comportement malveillant est réalisée en interceptant seulement une seule opération : IRP_MJ_WRITE. Nous souhaitons ainsi démontrer qu'il est possible de détecter les logiciels de rançon en utilisant moins d'indicateurs que les contre-mesures précédentes. Celles-ci utilisent au moins trois fois plus d'indicateurs. Nous interceptons également les opérations de type : IRP_MJ_SET_INFORMATION. Mais il s'agit uniquement d'empêcher la manipulation des fichiers et de leurs attributs une fois un comportement malicieux détecté. Ces opérations n'interviennent pas dans l'analyse comportementale. Les requêtes que nous filtrons sont interceptées en amont du pilote du système de fichiers (i.e., "preoperation callback"). Nous pouvons ainsi bloquer ou suspendre une opération suspecte avant qu'elle ne touche le système de fichiers. De plus, la détection d'un comportement malveillant se fait avec la granularité d'un fil d'exécution. Ce choix de conception nous permet de bloquer un fil d'exécution malicieux injecté dans un processus légitime tout en assurant la disponibilité de l'application.

Pour chaque requête d'écriture, notre pilote introduit une latence qui correspond au temps passé dans celui-ci. Il est nécessaire de minimiser ce temps autant que possible. Pour cela, nous collectons uniquement les informations essentielles qui sont :

— contenu de la requête d'écriture,
— taille de l'écriture,
— position dans le fichier,
— nom absolu du fichier,
— nom du processus,
— identifiant du processus,
— identifiant du fil d'exécution.

Ces informations sont copiées dans une zone mémoire non paginée puis transmises à un fil d'exécution. Un nouveau fil d'exécution est donc créé pour chaque requête d'écriture, celui-ci ne fait pas parti de la pile du gestionnaire des I/O. En conséquence, les requêtes d'écriture une fois les informations copiées sont immédiatement autorisées à poursuivre leur chemin dans la pile. On espère préserver les performances. Cette approche désynchronisée nous permet de surveiller l'ensemble du système de fichiers mais aussi tous les fils d'exécution de l'espace utilisateur. En effet, ce sont les fils d'exécution ainsi créés qui vont analyser de manière asynchrone les opérations sur le disque. Ces fils d'exécution peuvent faire des calculs, accéder à des ressources partagées, etc, et ce sans contrainte temporelle. La FIGURE 5.2 présente l'architecture simplifiée de notre contre-mesure : Data Aware Defense.

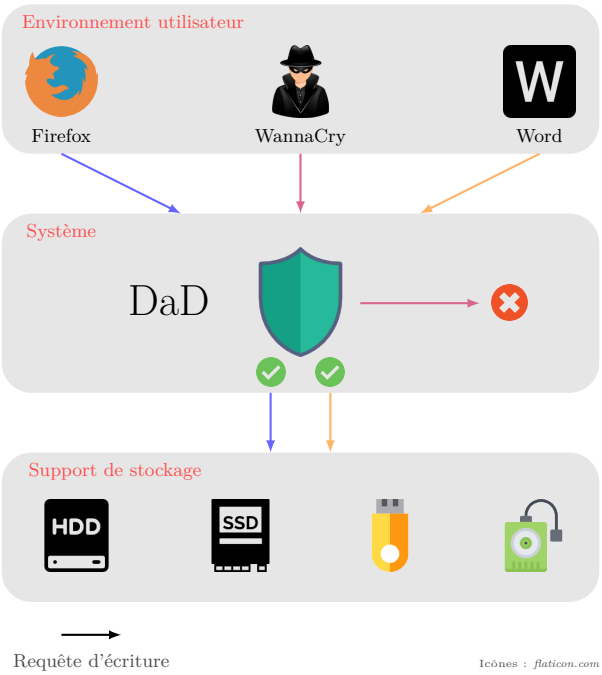

FIGURE 5.2 – Architecture de *Data Aware Defense* (DaD).

Une fois les indicateurs de compromission calculés, une structure interne est mise à jour. Celle-ci est en mémoire non paginée pour des raisons de performance et nécessite pour y accéder d'acquérir une ressource partagée. Cette structure qui est en fait une base de données, garde en mémoire l'historique récent de tous les fils d'exécution. Il s'agit de deux listes chaînées imbriquées l'une dans l'autre. La première liste énumère les processus, puis pour chaque processus celle-ci contient une liste de ses fils d'exécution avec les opérations correspondantes. Nous implémentons également notre propre ramasse miettes pour libérer la mémoire allouée dans notre base de données quand un fil d'exécution se termine. Enfin, pour analyser post-mortem le comportement des logiciels de rançon, notre pilote peut transmettre à une application dans l'espace utilisateur l'ensemble des informations surveillées dans une trace au format JSON. La partie suivante, présente deux indicateurs de compromission ainsi que des exemples de comportements de logiciels de rançon.

5.2 Les indicateurs de compromission

Dans cette partie, nous présentons deux indicateurs de compromission qui permettent de caractériser les logiciels de rançon. La combinaison de ces indicateurs est évoquée chapitre 6. Nos indicateurs ont les caractéristiques suivantes :

1. rapide,
2. simple,
3. interprétable,
4. ne nécessite l'interception que d'un seul type d'IRP.

De plus, aucun d'eux n'a précédemment été proposé dans la littérature [1] comme un indicateur de compromission des logiciels de rançon. Les données sont collectées sur la plateforme Malware - O - Matic, chapitre 3. Notre pilote fonctionne sur Windows 7 et 10, 32 et 64-bit. Les analyses sont réalisées de manière passive, c'est-à-dire sans altérer les requêtes.

5.2.1 Source d'entropie : test du Khi-deux

Une attaque d'un logiciel de rançon implique nécessairement un nombre important de requêtes d'écriture sur le système de fichiers au contenu chiffrés. Du moins, c'est le cas au moment de la rédaction de ce manuscrit. Pour détecter un tel comportement il est possible d'utiliser des tests statistiques. L'idée principale est que la distribution d'un chiffré est uniforme. Un test de qualité d'ajustement d'un échantillon (i.e., one sample goodness of fit test), calcule la distance entre une source d'information et une distribution théorique, que l'on appelle aussi le "modèle". En pratique, on compare un échantillon F avec une distribution connue G et l'on réfute ou non une hypothèse nulle. $H_0 : \forall x,\ F(x) = G(x)$. Par la suite, nous allons comparer nos échantillons avec la distribution aléatoire, soit la loi uniforme. Il ne s'agit pas de prouver que les deux ensembles proviennent d'une seule et même distribution, mais plutôt qu'il n'existe pas de différences significatives entre elles. L'objectif est de déterminer si un test est pertinent pour détecter une attaque de logiciel de rançon en temps réel.

Entropie de Shannon

Nous discutons de l'entropie de Shannon car elle est utilisée par toutes les contre-mesures basées sur l'activité du système de fichiers, c'est-à-dire [71, 103, 88]. Il s'agit d'une mesure de l'incertitude associé à une source d'information. Une grande diversité favorise une entropie élevée et des données structurées une entropie faible.

L'entropie d'une variable aléatoire discrète X définie sur l'alphabet $\Omega = \{x_1, x_2, \ldots, x_n\}$ avec sa fonction de distribution $p(x_i)$ est :

$$H(X) = -\sum_{i=1}^{n} p(x_i)\ \log\ p(x_i) \tag{5.1}$$

1. le χ^2 est utilisé comme indicateur par [40] indépendamment de nous

$H(X)$ est maximale quand l'occurrence de chaque symbole suit une loi uniforme. On s'intéresse ici à l'entropie d'une suite d'octets. Un octet peut prendre 256 valeurs possibles. La quantité d'information transmise est donc au maximum de $log_2(2^8) = 8$ bits par symbole ou caractère.

Test du Khi-deux

Le test du Khi-deux mesure la distance entre une série de valeurs, c'est-à-dire nos observations, et une loi de distribution connue. On parle de test d'ajustement du χ^2. Il s'agit d'un test non paramétrique, c'est-à-dire qu'aucune hypothèse n'est faite concernant la distribution des données. En effet, nous ne savons pas de quelle manière, le contenu des requêtes d'écriture est distribué dans un système. Pour chaque requête d'écriture le χ^2 est calculé.

Nous nous intéressons aux requêtes d'écriture avec la granularité d'un octet. Pour cela, les données observées sont considérées comme discrètes et rangées dans un histogramme $[\![0; 255]\!]$ avec un degré de liberté k égal à 255 (i.e., nombre de valeurs possibles moins 1). En supposant que N_i est le nombre d'occurrences observées pour une classe i, et que n_i est le nombre d'occurrences attendues pour la distribution testée. La formule pour calculer le Khi-deux est :

$$\chi^2 = \sum_i \frac{(N_i - n_i)^2}{n_i} \tag{5.2}$$

Une grande valeur indique que l'hypothèse nulle n'est probablement pas vérifiée, c'est-à-dire que les N_i ne peuvent être associées aux n_i. Le seuil de signification ou risque de première espèce du test α est la probabilité de rejeter l'hypothèse nulle alors que celle-ci est vraie. Nous choisissons un seuil : $\alpha = 5\%$. La statistique du χ^2 est comparée à une valeur de seuil appelée valeur critique notée z. Celle-ci est fonction du degré de liberté k, du seuil de signification α et bien entendu de la loi du χ^2. Si nous obtenons une statistique du χ^2 plus extrême que la valeur critique alors nous rejetons l'hypothèse nulle. La FIGURE 5.3 illustre notre test de l'hypothèse nulle, soit que la distribution des données observées et la loi uniforme est consistant, et la valeur critique $z_{5\%}$.

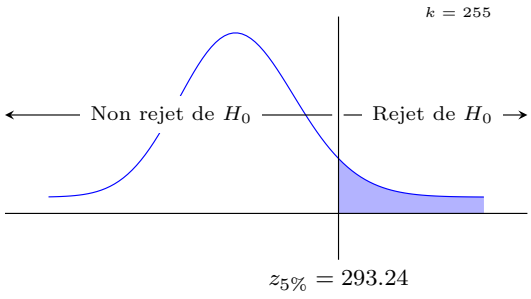

FIGURE 5.3 – Loi du χ^2 pour $k = 255$.

Une statistique inférieure à $z_{5\%}$ est interprétée comme une requête d'écriture chiffrée. En effet, nous testons l'adéquation d'une série d'observations avec la loi uniforme. La robustesse de

notre test doit être discutée dans des cas d'usage réels.

Comme la plupart des tests statistiques, le test du χ^2 donne des résultats inexacts pour des échantillons de petites tailles. Par convention, on considère qu'un échantillon est trop petit si le nombre d'occurrences attendues pour une classe est inférieur ou égal à 5. On peut raisonnablement penser que cela se produit souvent dans un système. La fiabilité du test augmente de pair avec la taille de l'échantillon. Un échantillon de petite taille est conservateur via-à-vis de l'hypothèse nulle. Nous faisons le choix de calculer le χ^2 même pour de petites échantillons. Nous favorisons ainsi les faux positifs. Nous limitons le nombre d'octets copiés puis impliqués dans le calcul à 10Ko. Il est nécessaire de fixer une limite, pour des raisons de performances, mais aussi pour éviter que notre contre-mesure ne soit victime d'un déni de service (e.g. zip bomb [17]). Le χ^2 n'est pas employé dans la littérature par des contre-mesures aux logiciels de rançon, mais il s'agit d'une statistique utilisée dans de nombreuses applications. On peut citer à titre d'exemple, *ENT* [1] et *BinWalk* [14].

Observations

Mbol *et al.* [96] démontrent que l'entropie de Shannon n'est pas un bon indicateur pour distinguer des images JPEG et des fichiers chiffrés. Le format JPEG contient des données compressées. La taille d'un fichier une fois compressé diminue mais pas la quantité d'information qu'il contient. Le nombre de bits par symbole augmente donc. C'est pourquoi il est difficile de discerner des fichiers chiffrés et compressés avec l'entropie de Shannon, car ceux-ci possèdent une entropie élevée. Le χ^2 au contraire est un outil statistique plus pertinente, car plus sensible. FIGURE 5.4, il est possible de distinguer des images JPEG (•) et des chiffrés (•) avec le χ^2. La tâche est moins triviale avec l'entropie de Shannon.

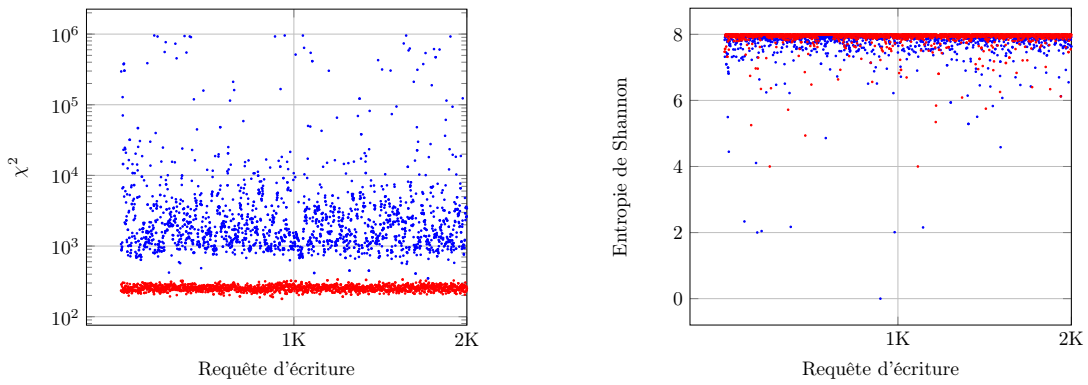

FIGURE 5.4 – χ^2 et entropie de Shannon : (•) rotation de JPEG (•) chiffrement.

Le TABLEAU 5.1 indique que d'autres formats de fichiers peuvent également déclencher des faux positifs avec l'entropie de Shannon. Le TABLEAU 5.2 montre que le χ^2 pour les mêmes fichiers rejette à raison avec une grande confiance l'hypothèse nulle. Ces observations font du Khi-deux notre premier choix pour détecter des opérations de chiffrement tout en limitant les faux positifs.

Tableau 5.1 – Entropie de Shannon pour 10Ko de fichiers de chaque type.

Type de fichiers	Minimum	Moyenne	Maximum	Variance
PNG	0.14	7.87	7.99	0.33
PDF	1.45	7.74	7.99	0.16
ZIP	3.21	7.93	7.99	0.07

Tableau 5.2 – Khi-deux pour 10Ko de fichiers de chaque type.

Type de fichiers	Minimum	Moyenne	Maximum	Variance
PNG	275.72	1.69e+6	3.76e+9	2.74e+15
PDF	306.86	1.50e+6	5.07e+8	1.30e+14
ZIP	220.44	4.74e+5	9.11e+8	1.23e+14

Nous vérifions également que le χ^2 est un bon indicateur pour détecter des requêtes d'écriture malveillantes. La FIGURE 5.5 illustre le comportement d'un échantillon de la famille *TeslaCrypt* sur le système de fichiers. Nous avons remarqué que ce comportement, qu'on peut qualifier d'usuel est commun à de nombreux logiciels de rançon. En effet, ceux-ci ne se contentent pas de chiffrer de manière agressive les fichiers utilisateur. Ils ajoutent bien souvent des notes de rançon dans les répertoires explorés, mais aussi des métadonnées aux fichiers chiffrés. Ces différentes opérations créent des "couches" bien visibles FIGURE 5.5. Ce comportement semble caractéristique des logiciels de rançon, bien que certains n'écrivent qu'une seule note de rançon sur le bureau. Dans ce dernier cas, le comportement malicieux d'un fil d'exécution reste identifiable.

La FIGURE 5.6 présente une trace d'exécution d'un logiciel de rançon, c'est-à-dire l'ensemble des requêtes d'écriture collectées pendant une attaque. On distingue bien les requêtes contenant des données chiffrées (•), celles-ci ont une statistique du χ^2 inférieure à $z_{5\%}$. Le reste des requêtes (•) ont des statistiques d'un ordre de grandeur bien supérieur. J'ai sciemment sélectionné un logiciel de rançon avec un comportement différent de la FIGURE 5.5. Les requêtes d'écriture du fil d'exécution malicieux ne sont pas décomposées en "couches" mais reste facilement identifiables comparé à l'activité courante du système.

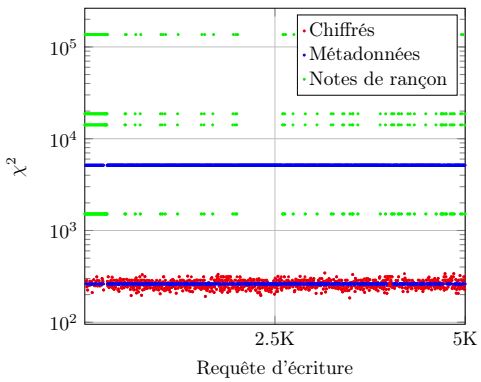

FIGURE 5.5 – Comportement d'un échantillon de la famille *TeslaCrypt*.

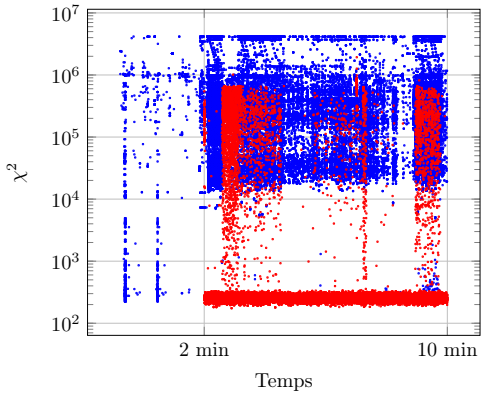

FIGURE 5.6 – Logiciel de rançon (•) et le reste du système en (•).

En conclusion, le test du χ^2 est préféré à l'entropie de Shannon car plus sensible et moins susceptible de déclencher des faux positifs. Le comportement des logiciels de rançon sur le système de fichiers est bien identifiable. La gestion des faux positifs est présentée dans le chapitre 6.

5.2.2 Comportement déviant : chaîne de Markov

Dans cette partie, nous proposons de modéliser par apprentissage des en-têtes de fichiers. Nous souhaitons ainsi représenter le comportement normal du système, c'est-à-dire les en-têtes manipulés par des applications légitimes. Lors du chiffrement des fichiers utilisateur les en-têtes sont remplacées par des données aléatoires. Ce comportement est caractéristique des logiciels de rançon. Les en-têtes de fichiers sont structurés et respectent des formats standardisés. Un apprentissage supervisé des en-têtes les plus courants est réalisable. La vraisemblance de chaque en-tête est ensuite déterminée. Si celle-ci est trop faible, l'en-tête est suspect. Une chaîne de Markov est utilisée comme modèle d'apprentissage.

Modélisation des en-têtes de fichiers

Nous considérons chaque requête en écriture comme une suite de variables aléatoires $X_n,\ n \in \mathbb{N}$, c'est-à-dire un processus stochastique. Une chaîne de Markov permet de modéliser l'évolution dynamique d'un tel processus. L'état du processus à l'instant n (i.e., temps) est représenté par un symbole X_n choisi dans un alphabet fini E de N éléments. Une chaîne de Markov vérifie la propriété de Markov, c'est-à-dire que son état futur ne dépend du passé qu'au travers de son état courant. Plus simplement, la transition vers un état futur ne dépend que de l'état présent, et non de ses états antérieurs. Une matrice stochastique $P(X_n, X_{n+1})$ donne les probabilités de transition.

Définition 5.1 *La probabilité de transition pour passer de l'état x à l'état y est la probabilité :*

$$P(x, y) = \mathbb{P}(X_{n+1} = y | X_n = x). \tag{5.3}$$

Définition 5.2 *Une matrice $P = (P(x, y),\ x, y \in E)$ est dite matrice stochastique si ses coefficients sont positifs et la somme sur une ligne des coefficients est égale à 1 :*

$$\forall\ x, y \in E \quad P(x, y)\ \geq\ 0 \quad et \quad \forall\ x\ \in E \quad \sum_{y\ \in\ E} P(x, y)\ =\ 1. \tag{5.4}$$

Définition 5.3 *Soit P une matrice stochastique sur E. Une suite de variables aléatoires $X_n,\ n \in \mathbb{N}$ à valeurs dans E est appelée chaîne de Markov d'ordre 1 de matrice de transition P si $\forall\ x_0, ..., x_n \in E$, tels que $P(X_0 = x_0, ..., X_n = x_n) > 0$:*

$$\mathbb{P}(X_{n+1} = x \mid X_0 = x_0, ..., X_n = x_n) = \mathbb{P}(X_{n+1} = x \mid X_n = x_n) = P(x_n, x). \tag{5.5}$$

Une chaîne de Markov est d'ordre $r > 0$ si sa mémoire est limitée aux r états passés, soit :

$$\mathbb{P}(X_{n+1} = x \mid X_0 = x_0, ..., X_n = x_n) =\ \mathbb{P}(X_{n+1} = x \mid X_{n-r+1} = x_{n-r+1}, ..., X_n = x_n). \tag{5.6}$$

Une chaîne de Markov est d'ordre $r = 0$ si le processus stochastique est sans mémoire.

Définition 5.4 *Une chaîne de Markov est homogène si pour tout $n \geq 0$, x et y dans E :*

$$\mathbb{P}(X_{n+1} = x \mid X_n = y) = \mathbb{P}(X_1 = x \mid X_0 = y). \tag{5.7}$$

Nous considérons ici uniquement des chaînes de Markov homogène.

Proposition 1 *La loi d'une chaîne de Markov homogène (CMH) est entièrement déterminée par sa distribution initiale π_0 et sa matrice de transition P. De plus, pour tous $n \in \mathbb{N}^*$, $x_0, ..., x_n \in E$:*

$$\mathbb{P}[(X_0, X_1, ..., X_n) = (x_0, x_1, ..., x_n)] = \pi_0(x_0)\ P(x_0, x_1)\ \cdots\ P(x_{n-1}, x_n) = p_n(x). \tag{5.8}$$

La probabilité π_0 est appelée loi initiale de la chaîne. Celle-ci correspond à la distribution initiale des différents états X_0. La matrice P^n est appelée matrice de transition en n pas. La loi de la chaîne de Markov X_n est le produit matriciel noté : $\pi_n = \pi_0\ P^n$.

Une matrice de transition P peut-être représentée par un graphe G dont les nœuds sont les états de E. Une arête est orientée de x vers y si et seulement si $P(x, y) > 0$. La FIGURE 5.7 illustre ainsi la trajectoire de notre chaîne sur les premiers bits d'un document Word. À noter qu'aucun état de notre chaîne n'est absorbant, c'est-à-dire tel que $P(x, x) = 1$.

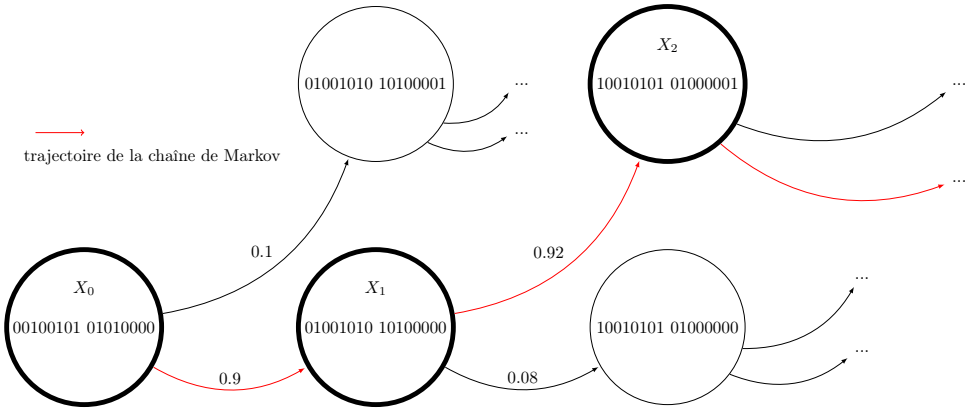

FIGURE 5.7 – Notre chaîne de Markov sur les premiers bits d'un document Word.

Nous nous intéressons aux requêtes d'écriture avec la granularité d'un bit. Notre modèle probabiliste considère donc un état de notre chaîne comme une suite de bits. Il faut alors déterminer le nombre d'états de notre chaîne qui va de pair avec la taille de sa mémoire. Le contexte dans lequel nous souhaitons utiliser note chaîne limite la taille de sa mémoire. En effet, notre chaîne est utilisée en temps réel par notre contre-mesure pour estimer la vraisemblance d'une opération. Celle-ci doit donc être présente en mémoire vive idéalement non paginée pour des raisons de performance. Nous ne pouvons pas garder en mémoire plus d'une dizaine de bits. De manière arbitraire on fixe l'ordre de notre chaîne $r = 16$, soit 2^{16} états. Nous évitons ainsi de faire du sur-apprentissage en utilisant 2 octets. Il s'agit d'un compromis entre précision et généralisation du modèle. Au moment $n = 0$, on se place dans l'état initial X_0 puis en fonction du bit lu on

saute vers un état X_1 et ainsi de suite. Chaque transition est donc une épreuve de Bernoulli de paramètre p. Une chaîne d'ordre $r = 32$, dispose d'un modèle qui consomme au moins 2^{32} bits. Il n'est pas possible pour un pilote de garder 500Mo en mémoire vive non paginée. À l'inverse, la mémoire consommée est de l'ordre de grandeur de 8Ko pour $r = 16$, ce qui est acceptable. Nous ne prenons pas en compte la loi initiale de la chaîne dans le calcul de la vraisemblance, celle-ci est omise sciemment, car elle n'a que peu d'influence pour de longues trajectoires. L'élément $\pi_0(x_0)$ dans l'équation (5.8) est remplacé par une constante multiplicative, c'est-à-dire 1.

Les en-têtes sont de tailles variables en fonction des types de fichiers, de leurs contenus, mais aussi des applications qui les génèrent. Nous réalisons un apprentissage sur les 2048 et 8192 premiers bits d'un fichier. Les résultats sur l'ensemble de test, nous permettrons de choisir la taille la plus pertinente. Nous estimons la fonction de transition de notre chaîne de Markov avec un corpus d'apprentissage de fichiers sains, en annexe B. Le corpus contient 45 extensions différentes et plus de 190k fichiers. Les fichiers les plus communs (e.g., doc, pdf) sont présents en plus grand nombre. La probabilité de passage de l'état x vers l'état j est égale au nombre de transitions de x vers y dans notre ensemble d'apprentissage divisé par le nombre de transitions dans l'état x.

Apprentissage et test de notre chaîne de Markov

Les résultats de nos deux modèles d'apprentissage sur l'ensemble de test sont visibles avec leurs matrices de confusion sur les FIGURES 5.8 et 5.9. N correspond au nombre de bits observés. L'ensemble de test est également visible en annexe B et contient plus de 190k fichiers.

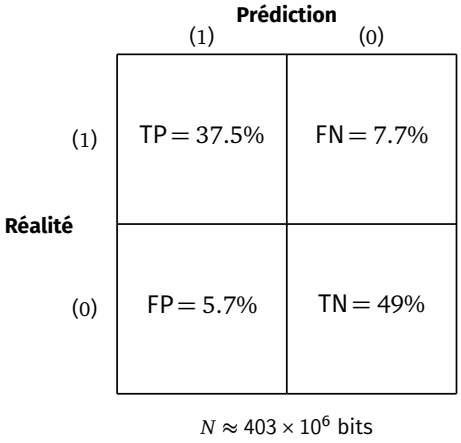

FIGURE 5.8 – Matrice de confusion de la chaîne de Markov sur l'ensemble de test. Celle-ci considère les 2048 premiers bits d'un fichier.

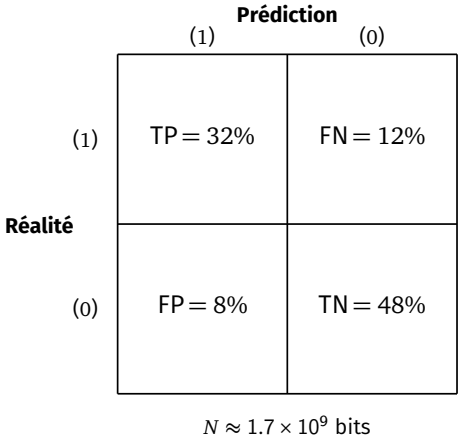

FIGURE 5.9 – Matrice de confusion de la chaîne de Markov sur l'ensemble de test. Celle-ci considère les 8192 premiers bits d'un fichier.

Les résultats sont meilleurs pour le modèle qui considère les 2048 premiers bits. L'apprentissage est donc restreint aux 2048 premiers bits. Ainsi, nous nous assurons que notre modèle n'apprend pas le contenu des fichiers, quitte à ce que celui-ci ait une vision partielle des en-têtes

les plus grands. En effet, l'apprentissage sur 8192 bits est moins pertinent, car le contenu de certains types de fichiers est sans doute inclus dans l'apprentissage. Ce qui biaise les résultats.

Nous utilisons ensuite trois métriques pour évaluer la qualité de notre modèle :

$$Justesse = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.9a}$$

$$Précision = \frac{TP}{TP + FP} \tag{5.9b}$$

$$Rappel = \frac{TP}{TP + FN} \tag{5.9c}$$

La justesse, équation (5.9a), est en fait la proportion de prédictions correctes. La précision, équation (5.9b), est la proportion de prédictions positives correctes, en effet le modèle peut engendrer des faux positifs. Le rappel, équation (5.9c), est la proportion de prédictions réellement positives correctement identifiée.

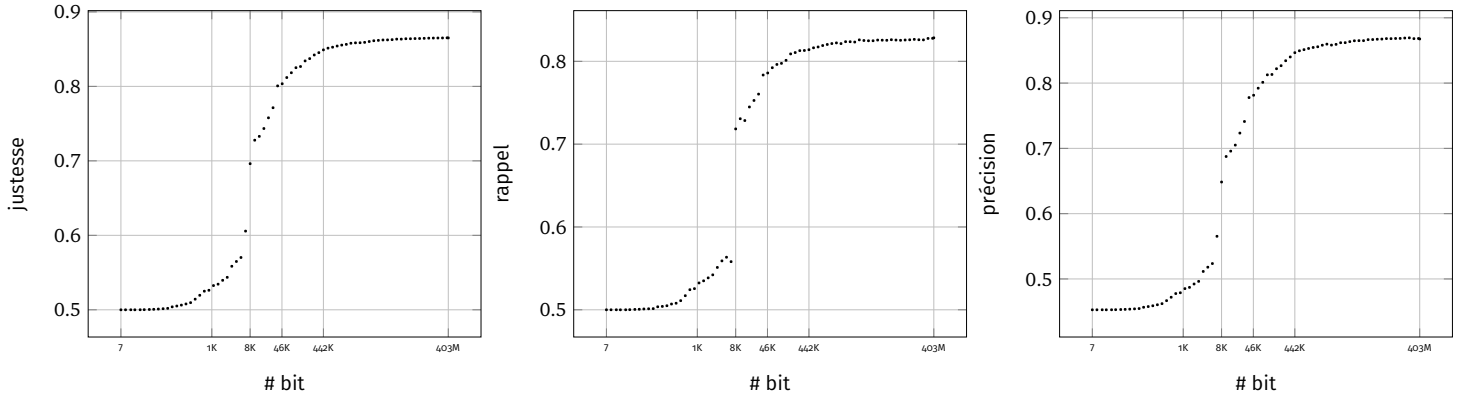

FIGURE 5.10 – Justesse, rappel et précision de notre chaîne de Markov sur l'ensemble de test en fonction du nombre de bits considérés dans l'ensemble d'apprentissage. À partir de 442Ko, la classification ne s'améliore que très peu.

La FIGURE 5.10 montre que notre chaîne de Markov sans mémoire, vérifie bien la loi de Bernoulli de paramètre $p = 0.5$. La classification commence à s'améliorer à partir de 8Ko observés. Finalement, une fois 442Ko observés dans l'ensemble d'apprentissage, la qualité de la classification ne progresse presque plus. Nous obtenons 86% de prédictions correctes, ce qui correspond au taux de justesse. Parmi le nombre de bits à 1 prédit, 82% c'est avéré vrai, ce qui est en fait le taux de précision. Enfin, notre modèle à un taux de faux négatifs de 7.7%, c'est-à-dire un bit à 1 prédit comme un 0. Ici, les faux négatifs n'ont pas plus d'importance que les faux positifs. Un bit à 1 n'est pas plus suspect qu'un bit à 0, il s'agit d'évaluer la vraisemblance globale d'une source Markovienne. Les performances de notre modèle nous semblent satisfaisantes. Nous n'investiguons donc pas plus : l'ordre de la chaîne ainsi que la taille des en-têtes.

Test du rapport de vraisemblance

Le test du rapport de vraisemblance, équation (5.10), permet de tester deux hypothèses simples. Il s'agit de comparer les probabilités d'une observation, c'est-à-dire une séquence de bits dans notre cas, selon deux modèles. Pour cela, nous utilisons la vraisemblance, telle que présentée équation (5.8). Le meilleur modèle est celui pour lequel l'observation est la plus vraisemblable. Nous comparons ainsi le modèle logiciel de rançon (i.e., aléatoire) et le modèle réel (i.e., notre chaîne de Markov). Le modèle aléatoire est une loi de Bernoulli de paramètre $p = \frac{1}{2}$.

Définition 5.5 *Soit $X_1, \ldots, X_n$ un échantillon de la loi P. On souhaite tester :*

$$\mathcal{H}_0 \ : P = P_0 \quad \textit{contre} \quad \mathcal{H}_1 \ : \ P = P_1 \ , \tag{5.10a}$$

$\mathcal{L}_0(X_1, \ldots, X_n)$ est la vraisemblance de l'échantillon sous $\mathcal{H}_0$, c'est-à-dire le comportement normal du système. $\mathcal{L}_1(X_1, \ldots, X_n)$ sa vraisemblance sous $\mathcal{H}_1$, le modèle aléatoire. Alors :

$$T = \frac{\mathcal{L}_1(X_1, \ldots, X_n)}{\mathcal{L}_0(X_1, \ldots, X_n)} \ , \tag{5.10b}$$

On appelle test du rapport de vraisemblance, le test défini par la règle de décision :

$$\textit{Rejet de } \mathcal{H}_0 \iff T > c \ . \tag{5.10c}$$

où une valeur de seuil c sous l'hypothèse $\mathcal{H}_0$ doit être déterminée expérimentalement.

Lemme 5.1 *Le lemme de Neyman-Pearson, nous dit que lorsque que l'on effectue un test d'hypothèse entre deux hypothèses $\mathcal{H}_0$ et $\mathcal{H}_1$, pour un échantillon $X_1, \ldots, X_n$ de P, que le test du rapport de vraisemblance, qui rejette $\mathcal{H}_0$ en faveur de $\mathcal{H}_1$, est le test le plus puissant de niveau α :*

$$P\left(\frac{\mathcal{L}_1(X_1, \ldots, X_n)}{\mathcal{L}_0(X_1, \ldots, X_n)} > \ c \mid P = P_0\right) \ = \ \alpha. \tag{5.11}$$

Il s'agit ici de régler le taux de faux positifs, c'est-à-dire le risque de première espèce, noté α. En effet, on souhaite marquer le moins souvent possible une requête d'écriture bénigne comme suspecte. De plus, le taux de vrais positifs est le meilleur possible d'après le lemme de Neyman-Pearson pour un taux de faux positifs choisi. Ce qui signifie, que la probabilité de rejeter l'hypothèse nulle à raison, c'est-à-dire de détecter une requête d'écriture suspecte, est la meilleur possible pour une probabilité α. La partie suivante illustre le fonctionnement du test du rapport de vraisemblance avec des logiciels de rançon. Le risque α et la valeur de seuil c correspondante, sont déterminés dans le chapitre suivant.

Observations

La FIGURE 5.11 présente une trace d'attaque d'un logiciel de rançon obtenue sur la plateforme MoM. Les fils d'exécutions malveillants ont été marqués manuellement dans les traces JSON. L'échantillon est particulièrement agressif (•). Les en-têtes de fichiers sont réécrits avec des données peu probables par rapport au modèle du comportement normal. Le ratio est largement en faveur du modèle aléatoire. Le reste des requêtes d'écritures (i.e., non malicieuses) pendant cette capture possède des scores bien distincts (•).

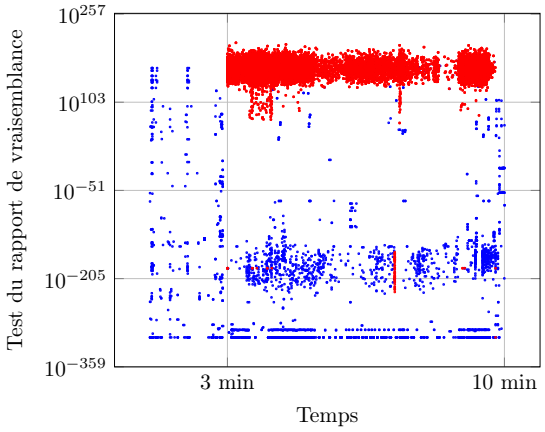

FIGURE 5.11 – Trace d'une attaque sur le système de fichiers avec le test du rapport de vraisemblance. Les requêtes d'écriture du logiciel de rançon (•) et le reste en (•).

La FIGURE 5.11 montre l'ensemble des requêtes en écriture du système pendant une attaque, les trois figures suivantes ne s'intéressent qu'aux fils d'exécutions identifiés comme malveillants. La FIGURE 5.12 illustre le fonctionnement d'un échantillon de la famille Cerber. Le comportement malveillant n'est pas capturé. En effet, cette famille ne s'intéresse pas au 512 ou 640 premiers octets d'un fichier, en fonction de la version de l'échantillon [5]. Les requêtes d'écritures capturées sont toutes très favorables au modèle réel. En fait, il s'agit des notes de rançon (•).

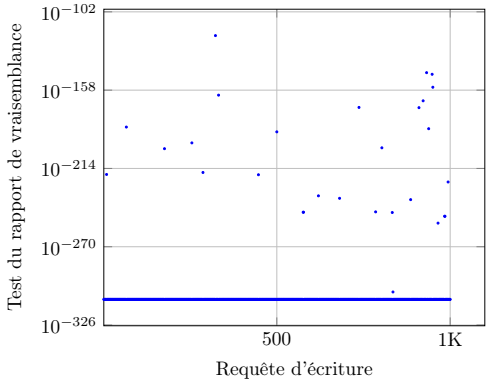

FIGURE 5.12 – *Cerber* sur le système de fichiers avec le test du rapport de vraisemblance.

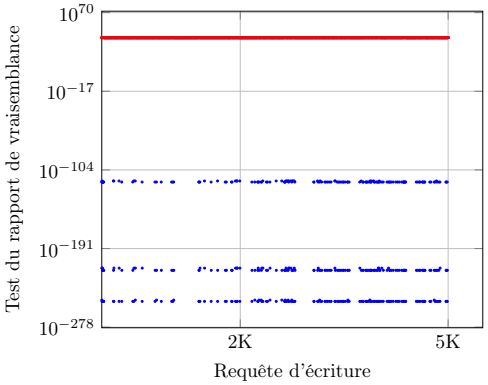

FIGURE 5.13 – *TeslaCrypt* sur le système de fichiers avec le test du rapport de vraisemblance.

FIGURE 5.13, un échantillon de la famille TeslaCrypt est présenté. Le comportement malveillant est capturé (•), mais moins facilement exploitable tel quel. Ce sont les requêtes dont le score est proche de 10^{42} qui modifient les en-têtes. Ces scores sont peu déviants comparés à ceux de la FIGURE 5.11. L'explication est simple, la famille TeslaCrypt ajoute une structure de données au début des fichiers chiffrés [38]. Cette structure contient entre autre des clés publiques. Les scores restent favorables au modèle aléatoire mais ne sont pas si aberrants. Cela peut s'expliquer par le fait que des champs ne sont pas utilisés (i.e., 0s). La FIGURE 5.14 est plus favorable à notre approche. L'échantillon de la famille Locky est très agressif. Les en-têtes sont réécrits par des données chiffrées, dont les scores sont très déviants (•).

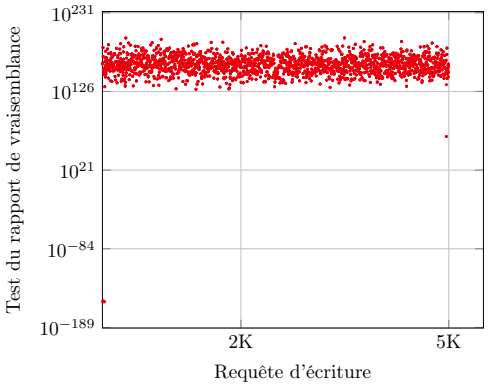

FIGURE 5.14 – Comportement d'un échantillon *Locky* avec le test du rapport de vraisemblance.

La FIGURE 5.15 compare le comportement de fils d'exécutions bénins. Firefox, Visual Studio, svchost et un fil d'exécution du système possèdent des requêtes dont le score est très déviant. La plupart des requêtes d'écritures sont favorables à notre modèle. Celui-ci ne peut être exhaustif, mais nous espérons qu'il capture la nature structurée ou non des données qu'il analyse. À noter que Kaspersky ainsi qu'un logiciel de sauvegarde, Crashplan, ne posent pas de problème.

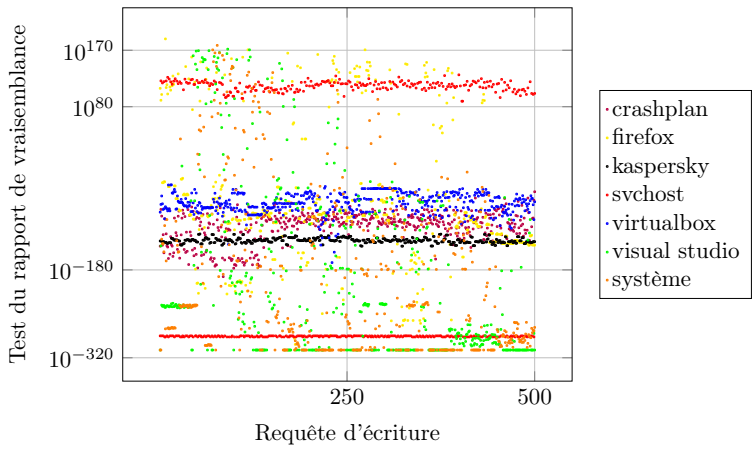

FIGURE 5.15 – Comparaison du comportement de fils d'exécutions bénins sur le système de fichiers avec le test du rapport de vraisemblance.

Il nous incombe dans la partie suivante de déterminer une valeur de seuil c ainsi que le risque α de première espèce correspondant pour que notre test du rapport de vraisemblance soit opérationnel. Le faible taux de faux positifs est un impératif. Nous notons déjà que cet estimateur n'est pas efficace contre tous les logiciels de rançon (e.g., Cerber). Le chapitre suivant identifie la proportion des échantillons dont nous capturons la nature malveillante.

5.3 Conclusion

Ce chapitre propose une solution générique pour observer le comportement des applications utilisateur sur le système de fichiers, et plus particulièrement, des logiciels de rançon. Cette solution est en fait un "File System Minifilter Driver", qui se positionne dans le noyau au plus près du système de fichiers. Celle-ci a l'avantage d'être difficile à détecter et désactiver pour un programme utilisateur malicieux. Les effets de bords sur le système ne sont pas simples à détecter (e.g., performance). Les artefacts présents dans le registre sont eux plus facile à découvrir, mais également à cacher si le besoin s'en fait sentir.

Nous proposons également deux indicateurs de compromission, simples et rapides, qui ne consomment que peu de ressources. En effet, ceux-ci nécessitent uniquement la surveillance des requêtes d'écritures, c'est-à-dire un seul type d'IRP. Le premier, le test du Khi-deux détecte l'écriture de nombreuses données aléatoires sur le système de fichiers avec la granularité d'un octet. Le second indicateur, est une chaîne de Markov qui représente les en-têtes courants manipulés par un utilisateur. Celle-ci est d'ordre 16 et s'intéresse à la distribution des 2048 premiers bits d'un fichier. Un test du rapport de vraisemblance est utilisé pour déterminer si une séquence de bits est plus favorable au modèle logiciel de rançon (i.e., modèle aléatoire) ou au modèle réel (i.e., notre chaîne de Markov). Ces deux indicateurs surveillent avec la granularité d'un fil d'exécution les opérations sur le système de fichiers.

L'efficacité de notre contre-mesure est discutée dans le chapitre qui suit. Celle-ci est vérifiée contre un large corpus de logiciels de rançon. De plus, nous collectons également des données utilisateurs bénignes. Le faible taux de faux positifs est un de nos objectifs. Ils nous faut donc calibrer notre contre-mesure.

Chapitre 6

Data Aware Defense : mise au point et évaluation

Dans ce chapitre, nous proposons deux contributions qui correspondent en fait à deux versions de notre contre-mesure appelée DaD. La première version, *Data Aware Defense (DaD) : Towards a Generic and Practical Ransomware Countermeasure* [PDB+17], a été présentée à la conférence *NordSec* en 2017. Celle-ci démontre qu'un taux de détection élevé peut être atteint avec un seul indicateur, tout en ayant un faible impact sur le système. La seconde version a fait l'objet d'un dépôt de brevet [1] en janvier 2018. Cette seconde itération s'accompagne de deux nouveaux objectifs : faible taux de faux positifs et restauration des fichiers infectés. Nous procédons donc graduellement pour adresser le problème des logiciels de rançon.

6.1 Une première itération : DaD v1.0

Nous reprenons ici l'architecture proposée au chapitre précédant, c'est-à-dire un "File System Minifilter Driver". Les requêtes d'écriture sont surveillées avec la granularité d'un fil d'exécution.

6.1.1 Preuve de concept avec un seul indicateur

Le test du Khi-deux seul est utilisé pour détecter un comportement malveillant. Nous nous limitons aux dix premiers kilo-octets d'une requête d'écriture pour calculer cette statistique. Une médiane glissante sur les 50 derniers scores du χ^2 est utilisée comme critère de décision pour marquer un fil d'exécution comme malveillant. La FIGURE 6.1 schématise notre test d'hypothèse nulle. L'objectif de cette statistique élémentaire est de capturer pour chaque fil d'exécution, la nature de son comportement sur le système de fichiers. Cela nous donne une idée générale de la distribution des requêtes d'écriture. Nous ne faisons aucune hypothèse concernant les motifs qui peuvent être observés sur le système de fichiers. De plus, cette statistique est très simple à calculer, ce qui nous permet de surveiller l'ensemble du système de fichiers. Ainsi, nous pouvons détecter une attaque au plus tôt, et l'utilisateur a moins de chance de perdre des fichiers importants.

1. n° de demande : 1850446 et n° de soumission : 1000444355

Le test d'hypothèse nulle :

Si :

Médiane du χ^2 sur les 50 dernières requêtes d'écriture $<$ valeur critique

Alors :

H_0 : Fil d'exécution malveillant

Sinon :

H_1 : Fil d'exécution non malveillant

FIGURE 6.1 – Test d'hypothèse de DaD avec le χ^2 pour détecter un fil d'exécution malveillant.

La plupart des logiciels de rançon écrivent de nombreuses notes de rançon dans chaque répertoire qu'ils explorent, afin d'obtenir le paiement de celle-ci. Ce comportement entraîne une hausse de la médiane glissante, notamment quand des répertoires qui contiennent peu de fichiers sont explorés. Le TABLEAU 6.1 montre des scores élevés du Khi-deux obtenus sur des notes de rançon. *In fine* le comportement malveillant est perceptible et détectable. La vitesse de détection et le nombre de fichiers perdus varient en fonction de la nature des répertoires infectés. Ce premier opus de DaD ne dispose pas d'un mécanisme de restauration des fichiers infectés. Le TABLEAU 6.2 donne des exemples de motifs sur le système de fichiers lors d'une attaque.

Tableau 6.1 – La statistique du Khi-deux pour les notes de rançon de deux échantillons.

Famille	Fichier	# octets	Khi-deux
TeslaCrypt	Recovery+jeexg.txt	2163	1.88×10^4
	Recovery+jeexg.html	9184	1.24×10^5
Cerber	# DECRYPT MY FILES #.txt	10510	1.59×10^5
	# DECRYPT MY FILES #.html	12381	1.81×10^5

Tableau 6.2 – Deux exemples des motifs exhibés sur le système de fichiers lors d'une attaque.

Famille	Opération	Fichier	# octets	Position	Khi-deux
TeslaCrypt	écriture	usurping.pdf	348	0	4894.11
	écriture	usurping.pdf	20	348	236.0
	écriture	usurping.pdf	6848656	368	268.87
Cerber	écriture	gursh.pdf	30198	34	271.28
	écriture	gursh.pdf	1	0, 1, .. 33	-
	écriture	gursh.pdf	52	30232	282.76
	écriture	gursh.pdf	72	30284	240.88
	écriture	gursh.pdf	256	30356	266.0

Nous fixons le risque de première espèce du test $\alpha = .05$, c'est-à-dire la valeur critique $z_{5\%} = 293.24$. Si nous obtenons en temps réel une médiane inférieure à la valeur critique, le fil d'exécution identifié comme malveillant est suspendu puis bloqué. Ce qui signifie que nous bloquons l'ensemble de ses accès en écriture sur le système de fichiers mais aussi les opérations de renommage. Nous collectons également des informations pour une analyse post-mortem. Pour cela, nous suspendons le processus du fil d'exécution malveillant, afin de récupérer l'image du binaire ainsi que l'ensemble des pages mémoires allouées puis touchées. Le processus est suspendu uniquement pendant le parcours de sa mémoire afin d'assurer la cohérence des données récupérées (e.g., code auto-modifiant). Nous préférons laisser le fil d'exécution malveillant s'exécuter sur la machine victime plutôt que le tuer et ainsi déclencher des mécanismes de défense. Une fois une attaque détectée puis bloquée, on peut imaginer que l'utilisateur averti du danger par notre contre-mesure éteigne son ordinateur dans l'attente d'une restauration.

6.1.2 Performance dans le monde réel

Un des points le plus important de notre contribution est d'adresser le problème des performances. Nous utilisons des outils standards pour réaliser les tests et les expérimentations sont reproductibles et comparables. En 2017, lors de la rédaction du papier, nous étions les seuls [2] pour une contre-mesure temps réel basée sur un pilote du système de fichiers à présenter des résultats satisfaisants qui permettent un déploiement dans le monde réel en production. Nous investiguons l'impact global de notre solution sur une machine utilisateur lambda [3]. Nous considérons le déploiement de DaD dans le cadre d'une utilisation professionnelle. Dans cette configuration comme nous allons le voir, DaD est presque imperceptible. Afin de se concentrer sur les performances lors des tests et de ne pas interférer avec les outils, nous avons désactivé le blocage d'un fil d'exécution. Le calcul de la médiane est toujours effectif, les mêmes calculs sont réalisés, seule la routine de gestion d'un fil malveillant est commentée.

Système de fichiers

Pour mesurer précisément l'impact sur le système de fichiers nous utilisons le Windows Performance Toolkit [44]. Il s'agit d'un outil conçu pour mesurer différentes métriques du système, notamment le temps consommé par les pilotes ajoutés à la pile du gestionnaire de filtre. Pendant plus de deux heures et demie nous enregistrons l'activité du système sur une machine dont l'utilisateur à une activité bureautique. Dans l'ensemble nous collectons 11348 requêtes d'écriture ce qui correspond au total à 133 ms dans notre pilote. DaD ajoute donc en moyenne un surcoût de 11.7 μs par requête d'écriture. À titre de comparaison, Scaife *et al.* [103] avec CryptoLock mesurent un surcoût de 9 ms par requête d'écriture. Même sans connaître le protocole de test de Scaife *et al.* [103], qui n'est pas donné, il est plus que probable que DaD améliore significativement cette performance par un facteur de plus de 100.

2. Kharraz *et al.* [88] présentent indépendamment Redemption
3. Windows 7 SP1 32-bit, Intel Xeon W3550, NVIDIA Quadro FX 1800, 4Go DDR3, SSD 120 Go SATA III

Nous utilisons par la suite deux outils standards : *CrystalDiskMark* [9] et *IOzone* [20]. Deux séquences de test sont à distinguer avec *CrystalDiskMark* : "sequential" et "random". La première réalise des écritures de grands blocs de données contigus sur le disque. La seconde génère de nombreuses requêtes d'écriture de petites tailles à des emplacements aléatoires sur le disque. Le TABLEAU 6.3 présente les résultats avec un SSD, nous observons que le test "random" entraîne une baisse significative de la bande passante. Il faut noter que la bande passante résiduelle, c'est-à-dire 9.5 Mo/s, est acceptable pour un usage professionnel. Sur un disque dur, les contraintes mécaniques du matériel brident les performances. C'est pourquoi, le test "random" serait borné par le matériel plutôt que notre pilote de filtre. En effet, le disque le plus rapide sur un site de benchmark [4] affiche une banque passante de 3.5 Mo/s au maximum et de 2.8 Mo/s en moyenne pour le test "random". Une très faible dégradation des performances est observée lors de l'écriture de gros blocs de données avec le test "sequential". Le test du Khi-deux ne s'intéresse qu'aux dix mille premiers octets d'une requête d'écriture, ce qui explique une consommation moindre des ressources du système dans ce dernier cas. En définitive, même dans le pire scénario (i.e., test "random"), le système de fichiers reste accessible et disponible.

Tableau 6.3 – *CrystalDiskMark* configuré pour 5 passes, taille des fichiers 2GiB, génération de données aléatoires et intervalle de 3 minutes. La bande passante du **SSD** est exprimée en Mo/s.

Test	DaD off	DaD on	Impact
Sequential	136.3	134.2	↘ − 1.5%
Sequential $Q32T1$	135.7	135.8	≃ + 0.07%
Random $4K$	55.28	9.581	↘ − 82.66%
Random $4K$ $Q32T1$	122.0	65.48	↘ − 46.32%

Kharraz *et al.* [88] présentent indépendamment en 2017 une contre-mesure avec un faible impact sur le système. La FIGURE 2.9 du chapitre 2 illustre les performances de REDEMPTION [88]. Ceux-ci utilisent *IOzone* comme outil d'évaluation. Nous avons donc réalisé une seconde série de tests avec *IOzone* dans des conditions similaires pour comparer les solutions. Une inconnue demeure, les auteurs ne précisent pas s'ils utilisent un HDD ou un SSD. Les TABLEAUX 6.4 et 6.5 présentent nos résultats. L'impact est quasi-nul avec un disque dur comme expliqué précédemment. Nous constatons avec un SSD une baisse des performances du même ordre de grandeur que celle induite par REDEMPTION [88].

Tableau 6.4 – Performance en Mo/s sur un **HDD** pour 100 x 512 Mo avec *IOzone*.

Test	DaD off	DaD on	Impact
Write	12.48	12.32	↘ − 1.28%
Rewrite	11.36	11.24	↘ − 1.05%

Tableau 6.5 – Performance en Mo/s sur un **SSD** pour 100 x 512 Mo avec *IOzone*.

Test	DaD off	DaD on	Impact
Write	54.68	50.78	↘ − 7.13%
Rewrite	53.71	52.73	↘ − 1.82%

4. http://hdd.userbenchmark.com/WD-Black-6TB-2015/Rating/3519

La baisse des performances est plus faible avec *IOzone* comparé à *CrystalDiskMark*. L'explication est simple, *IOzone* est paramétré pour écrire des fichiers de 100 Mo. Les deux tests ci-dessus ne génèrent pas une multitude de petits fichiers, ce qui est coûteux à surveiller. De plus, l'impact de REDEMPTION [88] est nécessairement plus important que DATA AWARE DEFENSE car il faut ajouter le surcoût induit par la surveillance des autres IRPs (e.g., lecture). Les tests réalisés TABLEAUX 2.9 et 2.10 ne permettent pas d'apprécier le coût réel de REDEMPTION.

CPU

En raison du grand nombre de fils d'exécution lancés par DaD pour effectuer des traitements asynchrones, nous réalisons des tests pour en estimer la charge sur le CPU dans différents scénarios. Deux outils standards sont utilisés : *Geekbench 4* [15] et *PCMark 8* [32]. Pour plus de détails concernant ces outils d'évaluation, veuillez vous référer à leur documentation. Nous mesurons initialement avec *Geekbench* un score de 6625 puis de 5841 après avoir activé DaD. Une baisse de performance de 11.83% est donc constatée. Avec *PCMark 8* le surcoût induit par DaD est moins palpable, le TABLEAU 6.6 présente les résultats. Deux scénarios sont joués, le premier se focalise sur des taches de bureautique, le second est orienté vers le divertissement (e.g., jeux). Une baisse des performances de moins de 1% est constatée dans les deux cas. L'impact de DaD sur le CPU est donc très minime.

Tableau 6.6 – Les scores de *PCMark 8*.

Test	DaD off	DaD on	Impact
Work	2859	2845	↘ − 0.49%
Home	2728	2705	↘ − 0.84%

Discussion

Dans cette section dédiée à l'évaluation de DaD nous n'investiguons pas l'impact sur la mémoire vive. Celui-ci peut être considéré comme négligeable. Nous notons environ 50 Mo de mémoire consommées lors d'un débogage noyau avec *Windbg* [43]. Mais cela peut varier en fonction de l'activité du système et du nombre de requêtes d'écriture que l'on souhaite garder en mémoire pour chaque fil d'exécution.

Une comparaison précise de DaD avec les contributions [103, 71, 88] est difficile. Les contributions ne disposent pas systématiquement d'une évaluation des performances. Quand celles-ci en ont une, la méthode est non-reproductible ou pas assez décrite. De plus, nous ne pouvons appliquer notre propre tests. Aucun des binaires des trois contributions du domaine n'est disponible. En effet, Continella *et al.* [71] ont répondu négativement à nos sollicitations et nous n'avons pas obtenu de réponse de Scaife *et al.* [103] et Kharraz *et al.* [88].

Nous notons un impact significatif introduit par DaD pour le test le plus plus exigeant (i.e., "random") quand celui-ci est réalisé sur un SSD. Néanmoins, notre contre-mesure est utilisable au quotidien comme le montre l'ensemble des tests. Cette contribution améliore significativement les travaux de [103, 71] d'un point de vue des performances. La contribution concurrente de Kharraz *et al.* [88] semble se positionner à un niveau de performance équivalent à DaD. Il aurait été intéressant de pouvoir le confirmer, mais en l'absence du binaire, c'est impossible.

6.1.3 Expérimentations et résultats

Les expérimentations sont basées sur des échantillons récupérés entre août 2016 et mars 2017 grâce à la plateforme Malware - O - Matic, comme expliqué au chapitre 3. Nous disposons ainsi de 798 logiciels de rançon actifs, visible en annexe A (i.e., campagne 1 et 2) et TABLEAU 6.7. Notre corpus est composé de plus de 20 familles, certaines sont présentes en très grand nombre, mais nous disposons aussi de singletons. L'étiquetage des échantillons est réalisé par *Avclass* [104].

Tableau 6.7 – Un aperçu des logiciels de rançon utilisés lors des expérimentations (i.e, 87.98%).

Famille	# échantillons	Famille	# échantillons	Famille	# échantillons
Teslacrypt	195 (24.43%)	Yakes	25 (3.13%)	Shifu	9 (1.12%)
Cerber	135 (16.91%)	Deshacop	19 (2.38%)	Fsysna	8 (1%)
Xorist	125 (15.66%)	Locky	17 (2.13%)	Shade	7 (0.87%)
Bitman	101 (12.65%)	Gpcode	13 (1.62%)	Dalexis	5 (0.79%)
Zerber	27 (3.38%)	Gamarue	9 (1.12%)	Usteal	5 (0.79%)

Notre corpus n'est pas limité par les échantillons qui utilisent des techniques anti-virtualisation grâce à l'emploi de MoM. Pour chaque échantillon, une analyse manuelle de son fichier journal au format JSON est réalisé. L'objectif est d'écarter les faux positifs (e.g., Vilsel) mais aussi de détecter les faux négatifs.

Détection des logiciels de rançon

DaD est intéressé uniquement par les requêtes en écriture avec la granularité d'un fil d'exécution indépendamment de toute signature. Notre contre-mesure est donc agnostique, ce qui est essentiel pour détecter des logiciels de rançon encore inconnus, c'est-à-dire de type "zero-day". DaD détecte avec succès 99.37% des logiciels de rançon, alors que 5 évadent la détection. La matrice de confusion FIGURE 6.2 présente les résultats, un seul faux positif est rencontré.

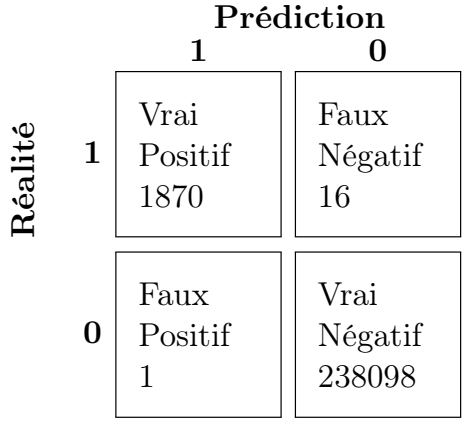

FIGURE 6.2 – La matrice de confusion pour les fils d'exécution surveillés par DATA AWARE DEFENSE lors de l'évaluation des échantillons : malveillant (1) et bénin (0).

Lors de l'analyse des échantillons une activité minimale est simulée sur la machine victime : déplacement de la souris, clavier et usage du navigateur. Cela explique en partie que seul un faux positif, un fil d'exécution de *Firefox*, soit rencontré. Au total plus de 238k fils d'exécution ont été surveillés pendant la phase de test. Le taux d'erreur de classification est très faible : 7×10^{-5}. Les processus courants du système qui comptent pour une bonne partie des fils d'exécution surveillés ne déclenchent pas de fausses alertes. Le comportement des logiciels de rançon est très particulier, il est donc peu probable que la médiane glissante du Khi-deux soit déclenchée par des processus bénins, à l'exception de certaines applications spécifiques.

Pour évaluer l'efficacité de DaD, la FIGURE 6.3 présente une estimation du nombre d'octets perdus pour l'ensemble des fils d'exécution malveillants. Il s'agit de la quantité de données infectées avant que DaD ne bloque le comportement malveillant. Nous constatons que pour 70% des fils d'exécution, c'est au plus 6.5 Mo qui sont perdus, ce qui reste acceptable pour la plupart des utilisateurs. Malheureusement, pour 90% des fils d'exécution l'utilisateur perd dans le pire scénario 70 Mo. En fonction des besoins des utilisateurs, une telle perte peut être tolérée, mais elle est inacceptable dans un contexte professionnel. À noter que la plupart des logiciels de rançon du corpus sont mono-fil d'exécution en ce qui concerne le chiffrement, respectivement 76.88% comme indiqué FIGURE 6.6. La FIGURE 6.4 présente la fonction de répartition des fils d'exécution malveillants, et pour chacun d'entre eux le nombre de requêtes d'écriture avant détection.

La détection est affectée par la nature des répertoires explorés, DaD peut être plus ou moins rapide pour identifier un comportement malveillant comme expliqué dans la partie 6.1.1. Même quand un tel scénario se présente, le fil d'exécution malveillant est détecté avec succès et l'utilisateur perd au plus une centaine de méga-octets. DaD détecte environ 30% des fils d'exécution après plus de 10k requêtes d'écriture. Ces fils d'exécution commencent par explorer à la racine *\Python27\Lib* qui contient une multitude de répertoires dans lesquels on ne trouve que quelques fichiers. Le fait de surveiller l'ensemble du système de fichiers sans distinction rend notre solution plus résiliente.

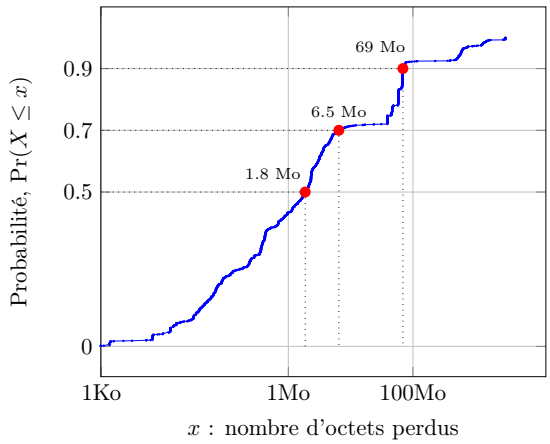

FIGURE 6.3 – La fonction de répartition des fils d'exécution malveillants, et pour chaque fil le nombre d'octets perdus.

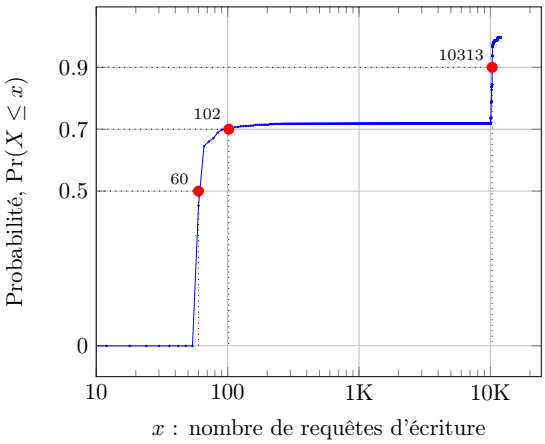

FIGURE 6.4 – La fonction de répartition des fils d'exécution malveillants, et pour chaque fil le nombre de requêtes d'écriture.

Notre heuristique, c'est-à-dire une médiane glissante du Khi-deux sur les 50 dernières opérations, est facile à contourner mais est néanmoins efficace. Nous n'investiguons pas plus le choix de ses paramètres : nombre de requêtes d'écriture à prendre en compte pour la médiane, risque de première espèce, taille minimale d'un échantillon, etc. Il suffit pour la contourner que moins de la moitié des requêtes d'écriture d'un fil d'exécution soient dédiées au chiffrement.

Une observation importante, déjà présentée dans le chapitre précédant FIGURE 5.5, est que nous pouvons distinguer différentes couches du Khi-deux (e.g., notes de rançon) sur le système de fichiers en fonction des données manipulées par un logiciel de rançon. Toutes les familles de logiciels de rançon ne "génèrent" pas systématique un tel comportement. Ces couches peuvent être utilisées comme un indice pour distinguer les logiciels de rançon qui implémentent ou non la routine de déchiffrement. En effet, l'ajout de métadonnées durant le processus de chiffrement est visible et suggère une chance de déchiffrer les données après le paiement. Cette observation peut aussi être utilisée pour distinguer une application légitime de chiffrement d'un logiciel de rançon. De plus, durant les expérimentations, nous n'avons observé aucune dissociation en plusieurs fils d'exécution des opérations suivantes : chiffrement, note de rançon et métadonnées.

Nous utilisons l'indice de diversité de Simpson, équation (6.1), pour évaluer la diversité des requêtes en écriture réalisées par les fils d'exécution malveillants. Cet indice est utilisé en écologie pour quantifier la diversité d'un habitat, il prend en compte le nombre d'espèces présentes ainsi que l'abondance de chacune de ces espèces. L'indice de Simpson mesure la probabilité que deux individus sélectionnés aléatoirement appartiennent à la même espèce. Plus la valeur de la statistique D est importante et plus la diversité est faible.

$$D = \frac{\sum n_i(n_i - 1)}{N(N-1)} \tag{6.1}$$

$$\begin{aligned} n_i &= \textit{nombre d'individus d'une espèce} \\ N &= \textit{nombre total d'individus dans l'échantillonnage} \end{aligned}$$

Nous remplaçons ici les espèces par le type des données manipulées en fonction de la statistique du Khi-deux : chiffré, note de rançon et métadonnées. Le résultat est visible FIGURE 6.5. L'indice est calculé uniquement sur les 50 dernières requêtes en écriture de chaque fil d'exécution malveillant. C'est-à-dire les opérations qui conduisent à l'identification du comportement malveillant. Nous observons ainsi que pour la moitié des échantillons, deux requêtes en écriture appartiennent à la même catégorie au minimum dans 58% des cas (i.e., peu de diversité). Pour 32% des fils d'exécution, c'est au plus dans 50% des cas. L'abondance de requêtes chiffrées donne plus de poids à cette catégorie, comparé à l'écriture des notes de rançon. Nous observons donc une diversité somme toute limitée. Seul 6% des fils d'exécution ont une statistique D entre 0.23 et 0.4. Le comportement en couches rapporté précédemment est néanmoins visible, sinon nous aurions des statistique plus élevées dans l'ensemble. Nous remarquons également que 20% des fils d'exécution ont un indice de Simpson supérieur à 0.8. Ces échantillons ne génèrent sans doute pas ou peu les couches du Khi-deux, l'essentiel de leur activité est dédiée au chiffrement.

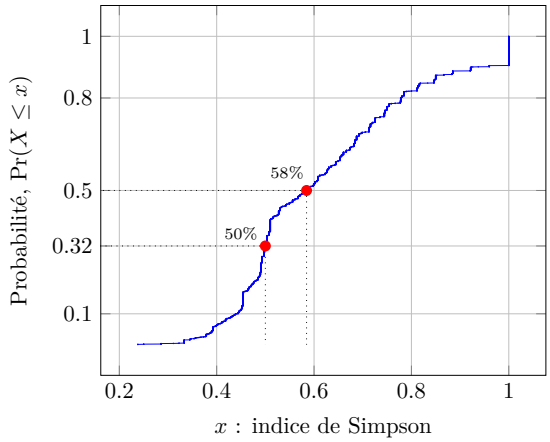

FIGURE 6.5 – La fonction de répartition des fils d'exécution malveillants, et pour chaque fil l'indice de diversité de Simpson.

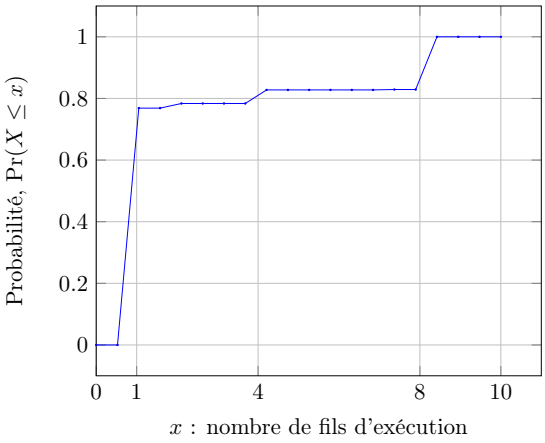

FIGURE 6.6 – La fonction de répartition des échantillons, et pour chaque échantillon le nombre de fils d'exécution malveillants.

Les résultats de nos expérimentations démontrent que DaD est une solution efficace pour stopper une infection en temps réel. Dans les parties suivantes, nous discutons des limitations, en particulier des faux positifs et négatifs.

Faux positifs

Le taux de faux positifs de DaD est 0.0004% si l'on considère les fils d'exécution surveillés pendant les expérimentations, FIGURE 6.2. Cependant, il est nécessaire de nuancer cette statistique extrêmement flatteuse qui n'est pas représentative du monde réel. Pour rappel, une activité minimale est simulée pendant les expérimentations. Il est difficile d'évaluer pertinemment les faux positifs, ceux-ci peuvent varier en fonction des utilisateurs, des applications utilisées et de l'usage que chacun fait d'une application. En l'absence d'un jeu de données étalon anonymisé, il est également difficile de comparer les résultats de chacune des contributions, car celles-ci ont des méthodes d'évaluation différentes. L'évaluation de cette statistique de manière statique est limitée, les stimuli d'un utilisateur sont plus représentatifs.

En choisissant un risque de première espèce α de 0.05 pour la médiane glissante du Khi-deux nous éliminons un nombre significatif de faux positifs parmi la plupart des applications bénignes. DaD surveille en temps réel des dizaines de processus et des centaines de fils d'exécution. Le fait de surveiller le système de fichiers avec la granularité d'un fil d'exécution nous permet de ne pas bloquer un processus tout entier, ce qui n'est pas négligeable en cas de faux positif. Seules des applications spécifiques sont capables de mimer le comportement des logiciels de rançon du point de vue de notre indicateur de compromission, le Khi-deux. Nous pouvons citer : compression, chiffrement, effacement sécurisé, etc. Notre indicateur est conçu pour détecter des opérations de chiffrement, il est donc normal que des applications bénignes déclenchent de fausses alertes. La FIGURE 6.7 présente des applications qui déclenchent des faux positifs à l'exception de *Mogrify* (●).

DaD n'est pas capable de distinguer la compression du chiffrement. Les applications *7-Zip* (□), *GPG4Win* (△) et *µTorrent* (○) déclenchent des faux positifs. *Mogrify* est utilisé pour faire pivoter des images JPEG, celles-ci sont intégralement réécrites sur le disque. Les statistiques du Khi-deux associées à *Mogrify* sont bien au-dessus de la valeur critique $z_{5\%}$ et ne déclenchent donc pas de faux positifs. Nous ne fournissons pas une liste exhaustive des applications qui déclenchent des faux positifs, là n'est pas notre objectif. Le Khi-deux seul n'est pas suffisant pour éliminer les faux positifs. Dans tous les cas, une observation majeure est à noter : seul une couche du Khi-deux est observée. Le modèle économique des logiciels de rançon est basé sur l'extorsion. Pour être payé, ceux-ci doivent rendre les notes de rançon le plus visible possible, d'où les couches du Khi-deux. Il s'agit d'une piste à explorer pour des travaux futurs.

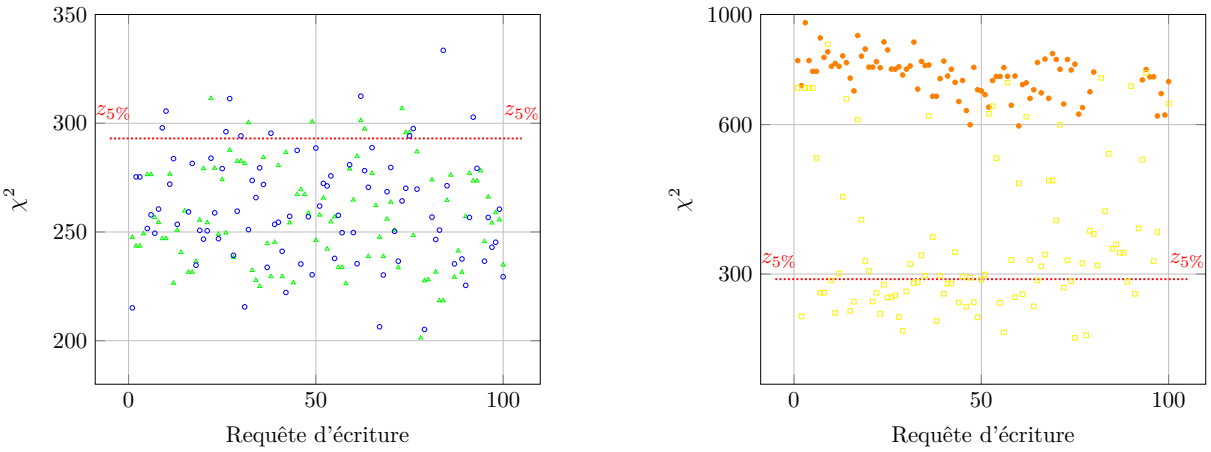

FIGURE 6.7 – La statistique du Khi-deux pour les 100 premières requêtes d'écriture des programmes suivants : *Mogrify* (●), *7-zip* (□), *GPG4Win* (△) et *µTorrent* (○).

Faux négatifs

Notre contre-mesure DaD s'intéresse à des opérations très spécifiques sur le système de fichiers qui appartiennent à des logiciels de rançon mais pas exclusivement. La nature des logiciels de rançon à ce jour est bien connue et documentée : ils chiffrent les fichiers. Cependant l'industrie des logiciels malveillants est très prolifique et nous ne sommes pas à l'abri d'un comportement plus furtif, il s'agit d'une course à l'armement. Par exemple, le problème spécifique du chiffrement de la section de démarrage n'est pas adressé ici, mais des solutions ont été proposées [39]. De plus, comme indiqué dans Mbol *et al.* [96] si un algorithme de chiffrement par substitution est utilisé, la distribution du fichier original est conservé, ce qui permet d'évader la détection car une source d'entropie non structurée est l'essence même de notre test du Khi-deux.

Les logiciels de rançon qui entrelacent chaque opération d'écriture contenant du chiffré par suffisamment d'opérations bénignes ne sont pas détectés. C'est-à-dire au moins 25 opérations non malicieuses pour une fenêtre glissante, cela suffit pour berner notre heuristique basée sur la médiane. La FIGURE 6.8 présente deux faux négatifs qui exploitent cette limitation.

Avant de prendre une décision concernant un fil d'exécution, nous devons observer au moins 50 requêtes en écriture valides. Un logiciel de rançon qui associe à chaque fichier chiffré un seul

fil d'exécution éphémère n'est pas détecté par DaD. Un tel comportement est néanmoins peu courant et peut alerter un antivirus. Un logiciel de rançon peut également chiffrer lentement, mais ca ne sert pas son objectif. Si un utilisateur se rend compte d'une attaque, il peut réagir. DaD n'est pas sensible à de telles attaques, le temps n'intervient pas lors de la prise de décision. Enfin, une faille dans le noyau inconnue ou non corrigée est une brèche potentielle qui peut être utilisée pour détecter puis désactiver DaD et même compromettre complètement le système.

Parmi notre corpus 16 fils d'exécution malveillants ne sont pas détectés, soit cinq logiciels de rançon pour 0.62% des échantillons. 3 échantillons de la famille *Xorist* utilisent des primitives cryptographiques peu sûres, la distance avec la distribution uniforme est trop importante pour être identifiée comme du chiffré. Les deux échantillons restant sont visibles FIGURE 6.8 et diffèrent de tout ce que nous avons précédemment observé.

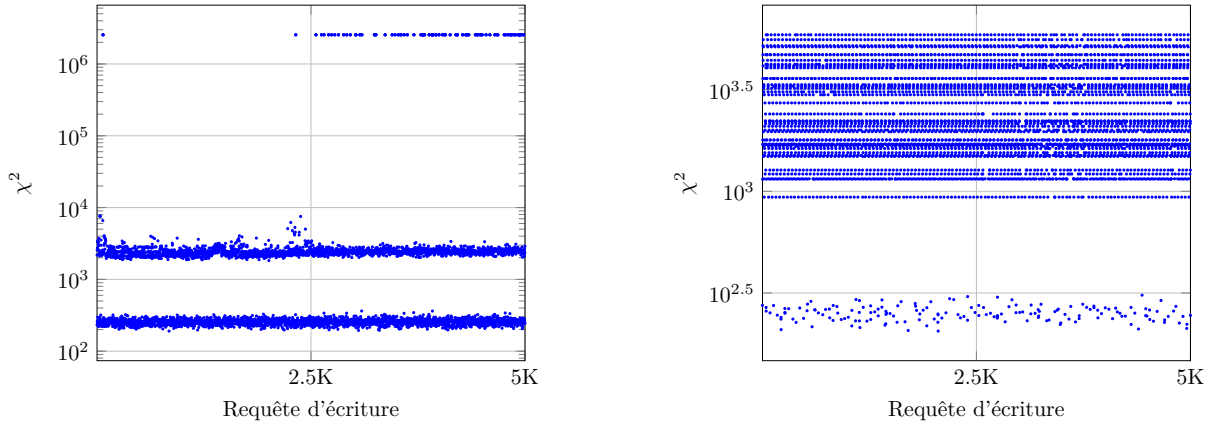

FIGURE 6.8 – Deux des cinq faux négatifs de notre corpus. À gauche un échantillon de la famille *Crysis* [5] et à droite de la famille *Purge* [6]. Ces deux échantillons entrelacent chaque opération d'écriture malveillante avec de nombreuses écritures bénignes.

L'échantillon *Crysis* n'écrit pas de notes de rançon et après environ 3k requêtes d'écriture commence à écrire de grands blocs de données de taille 2^{18} octets. Ces opérations ne contiennent que des zéros et sont répétées plusieurs fois. Elles sont donc inutiles et redondantes si ce n'est pour abuser une contre-mesure, telle que DaD par exemple. De plus, pour chaque requête de chiffrement sur un fichier sain, des métadonnées sont ajoutées. Le second échantillon *Purge* cache son comportement malveillant derrière une multitude de requêtes en écriture dont le contenu n'est autre chose que la note de rançon découpée en petits morceaux. Pour chaque fichier chiffré, la note de rançon est réécrite (i.e., 42 x 128 octets).

La médiane glissante est donc insuffisante pour détecter de tels comportements. Les auteurs ont la volonté manifeste de cacher le processus de chiffrement grâce à des opérations redondantes qui manipulent des données structurées. Nous pouvons parler d'attaques par mimétisme. L'objectif est de reproduire un comportement normal (i.e., texte) ou de faire baisser un niveau d'alerte (i.e., zéros), tout en procédant au chiffrement des données de l'utilisateur à la marge.

5. 5ceb475c11c88aaf2f0967090e78ef4b
6. 2297ef360a7efced9c8b6620059bbc99

6.1.4 Synthèse

Notre contre-mesure Data Aware Defense (DaD) est basée sur un pilote de type "File System Minifilter Driver" et surveille les requêtes d'écriture sur le système de fichiers. Celle-ci est disponible sur mon site académique[7]. Aucune hypothèse n'est faite concernant des motifs que les logiciels de rançon sont censés reproduire. De plus, nous traçons les requêtes sur le disque avec la granularité d'un fil d'exécution. Aucun répertoire n'est exclu de notre surveillance. Le test du Khi-deux comme indicateur est une bonne alternative à l'entropie de Shannon.

Notre contribution majeure est le fait que DaD introduit un surcoût raisonnable, et peut être déployée sur des Windows 7 ou 10 en production. Le surcoût estimé en moyenne pour une requête en écriture est de $11.7\mu s$, soit $\frac{9 \times 10^{-3}}{11.7 \times 10^{-6}} \simeq 750$ fois mieux que Scaife *et al.* [103]. Nos expérimentations montrent que les logiciels de rançon les plus évolués utilisent des attaques par mimétisme. Cependant nous avons détecté avec succès 99.37% des échantillons avec au plus 70 Mo perdus par fil d'exécution dans 90% des cas et moins de 7 Mo dans 70% des cas.

Tableau 6.8 – Évaluation de Data Aware Defense (DaD).

Corpus	Résultats
Nombre total d'échantillons	798
Nombre d'échantillons détectés	793 (99.37 %)
Faux négatifs	5 (0.62 %)
Faux positifs	1 (0.0004 %)

Les résultats obtenus sont encourageants que ce soit d'un point de vue des performances ou du taux de détection, tout cela grâce à l'utilisation d'une seule métrique calculée pour tous les fils d'exécutions de tous les processus utilisateur. Le tableau 6.8 résume brièvement les résultats.

Les faux positifs sont inhérents à notre approche car nous détectons des opérations de chiffrement sur le système de fichiers. Des applications légitimes peuvent donc déclencher une alerte. Le taux de faux négatifs est très faible et le calcul de notre heuristique la médiane du Khi-deux est rapide. Il s'agit donc d'une bonne première ligne de défense. Une fois qu'un fil d'exécution apparaît comme suspect, plutôt que de le bloquer, des métriques plus coûteuses pourraient être utilisées pour diminuer le taux de faux positifs sans impacter les performances.

Nous avons déjà souligné que dans des travaux futurs la reconnaissance des couches du Khi-deux, c'est-à-dire des chiffrés et des données constantes, peut être employée pour améliorer le taux et la vitesse de détection. L'interaction avec un utilisateur est à prévoir, mais seulement une fois que le nombre de faux positifs atteint un taux raisonnable. En effet, un utilisateur ne peut ignorer l'exécution d'une solution de chiffrement ou d'un outil de compression sur sa machine, il est donc capable d'identifier des modifications non souhaitées. Dans la partie suivante, nous ajoutons un second indicateur ainsi qu'un second niveau d'alerte, tout cela pour faire baisser le taux de faux positifs. Une collecte de données utilisateur nous permet également de mieux évaluer le comportement de DaD dans le monde réel et de mettre au point ses paramètres.

7. http://people.rennes.inria.fr/Aurelien.Palisse/DaD.html

6.2 Une seconde itération plus aboutie : DaD v2.0

Dans cette partie nous proposons une seconde version de DaD. Celle-ci dispose de deux indicateurs de compromission, le test du Khi-deux et la chaîne de Markov construite au chapitre 5. Les sections suivantes ont pour objectifs de mettre au point les paramètres de DaD : valeur de seuil, faux positifs, etc. Les paramètres sont déterminés *offline* grâce aux traces d'exécution obtenus avec MoM et sur la machine d'un utilisateur lambda. Il s'agit d'une extension de notre contribution [PDB+17] présentée à *NordSec* 2017.

6.2.1 Déploiement de DaD

DaD a été installé en mode "passif" sur la machine de travail d'un doctorant de l'équipe le 08/11/2017. La collecte des données a pris fin le 24/11/2017. Son ordinateur est équipé d'un Windows 10 64-bit. Aucun effet indésirable notable n'a été rapporté (e.g., ralentissement). Un écran bleu, c'est-à-dire une erreur fatale du système, est imputable à DaD sur les 17 jours de collecte. L'utilisateur peut à tout moment désactiver la collecte des données s'il le souhaite lors d'une session, les 17 jours de collecte ne sont donc pas plein. Au total nous avons collecté des données pendant 212 heures. À noter que son ordinateur est équipé de *Kaspersky*, ce qui ne semble pas poser de problèmes. On peut voir DaD comme une sonde, car celui-ci se contente de récupérer les données qu'il observe, sans jamais bloquer un fil d'exécution. Les données sont collectées par période de 10 minutes en RAM puis stockées sur la machine de l'utilisateur. Nous les récupérons ensuite périodiquement. La sonde intercepte l'ensemble des requêtes d'écriture provenant de l'espace utilisateur et calcule les statistiques suivantes :

— Le Khi-deux avec au maximum 2^{14} octets intervenant dans le calcul. Principalement pour des raisons de performance mais aussi pour éviter les attaques "zip bomb" [17]. La taille minimum d'un échantillon pour donner du sens au calcul est fixée arbitrairement à 2Ko.

— Le test du rapport de vraisemblance entre le modèle réel (notre chaîne de Markov) et le modèle logiciel de rançon (aléatoire). Pour cela, nous utilisons le modèle sain construit section 5.2.2 à partir des 2048 premiers bits des fichiers du corpus d'apprentissage. Nous comparons ensuite les probabilités qu'une séquence de bits de taille au moins 1024 contenu parmi les 2048 premiers bits d'un fichier appartienne davantage à l'un ou à l'autre modèle.

6.2.2 Couverture de notre contre-mesure

Le TABLEAU 6.9 représente les techniques utilisées ou qui peuvent l'être par les logiciels de rançon pour s'attaquer aux données de l'utilisateur. Les algorithmes de chiffrement par substitution et une faible vitesse ne sont pas implémentés à ce jour d'après nos connaissances. Un logiciel de rançon qui ne chiffre pas les en-têtes et qui utilise un algorithme de chiffrement par substitution monoalphabétique passe à travers notre contre-mesure. Un tel algorithme de chiffrement est peu résistant. La vitesse avec laquelle un logiciel de rançon s'attaque aux données est un facteur à étudier. 90% des logiciels de rançon collectés par Huang *et al.* [83] de 2016 à 2017 chiffrent les données utilisateur en moins de 5 minutes. Les attaques par ressemblance ou mimétisme peuvent également se développer. Deux échantillons exhibent un tel comportement FIGURE 6.8.

Tableau 6.9 – Ensemble non exhaustifs de techniques utilisées ou potentiellement utilisées par les logiciels de rançon pour attaquer le système de fichiers. ✓ pour les méthodes couvertes et ✗ sinon.

	Khi-deux	Test du rapport de vraisemblance
Avec en-tête	✓	✓
Sans en-tête	✓	✗
Chiffrement standard	✓	✓
Chiffrement par substitution	✗	✓
Faible vitesse	✗	✓
Attaque par mimétisme	✗	✗

6.2.3 Analyse des données collectées

Nous distinguons ici, les données issues de la machine de notre volontaire et celles provenant d'attaques de logiciels de rançon sur MoM.

Distribution des données : usage courant

Seul un membre de notre équipe de recherche CIDRE est équipé de DaD. Il est difficile de trouver des volontaires, une appréhension lié au déploiement d'un pilote bas niveau est notable. De plus, des contraintes techniques rendent son déploiement non triviale à Inria. En effet, il faut désactiver temporairement *Bitlocker* (i.e., installé par défaut) le temps de l'installation, pour cela il faut obtenir la clé "maître" auprès du service informatique. Nous ne disposons pas d'un certificat de signature donc il nous faut aussi désactiver une sécurité (i.e., activer le mode test) sur la machine de l'utilisateur, ce qui introduit une brèche de sécurité. Il aurait été intéressant de collecter des données de différentes versions de Windows mais aussi de plusieurs utilisateurs.

Les FIGURES C.1 et C.2 en annexe C permettent de visualiser l'évolution des indicateurs en fonction du temps sur une plage de 17 jours. Au total, 1.45 million de requêtes d'écriture ont été capturées pour le test du rapport de vraisemblance et 38 millions (i.e., 15 millions si 2Ko minimum) pour le test du Khi-deux. Au total le jeu de données fait 12Go pour les traces saines.

La FIGURE 6.9 présente la distribution de la statistique du Khi-deux. Si nous considérons l'ensemble des requêtes en écriture sans distinction (●) : 17% ont une statistique inférieure à la valeur critique $z_{5\%}$. Il est étonnant de voir qu'en ne sélectionnant que les requêtes de taille au minimum 2Ko (●) cette proportion augmente à 36%. En fait, le processus *Kaspersky* écrit périodiquement de gros blocs de données dont le contenu est très proche d'une distribution uniforme. La probabilité d'occurrence de ces données est inférieure à 1×10^{-5} pour bon nombre d'entre elles selon la loi du Khi-deux. Les statistiques obtenues par ce processus sont pour 83% d'entre elles inférieures à 200. Les données observées sont donc trop proches du "modèle aléatoire" pour être considérées valides. Il s'agit là sans doute d'une manipulation de *Kaspersky* pour protéger les données[8]. Celles-ci sont donc retirées du jeu de données. Nous pouvons faire un parallèle avec la controverse de Mendel. En 1866, Mendel [98] publie des résultats qui seront

8. \Device\HarddiskVolume3\ProgramData\Kaspersky Lab\KES10SP1\SysWHist\main.sdb

ensuite contestés par certain. Pour Fisher *et al.* [76], les résultats sont : "*too good to be true*". En effet, ceux-ci sont plus proches du modèle proposé que les résultats attendus par les lois naturelles. Il se pose alors la question de savoir, si ceux-ci ont été volontairement modifiés ou si un biais a été introduit par Mendel et ses assistants lors des expérimentations. Pires *et al.* [101] discutent de cette controverse et concluent qu'il n'y a pas de fraude délibéré mais qu'un biais lié au processus de sélection dans les expériences en est à l'origine. Ils nous semblent ici, que *Kaspersky* modifie les données de manière particulière et biaise donc notre test du Khi-deux, en tant que produit de sécurité nous pouvions nous y attendre. La fonction de répartition (●) est alors plus favorable à notre souhait de limiter les faux positifs. Nous observons ainsi 5% de faux positifs pour un total de 9.8 millions de requêtes en écriture.

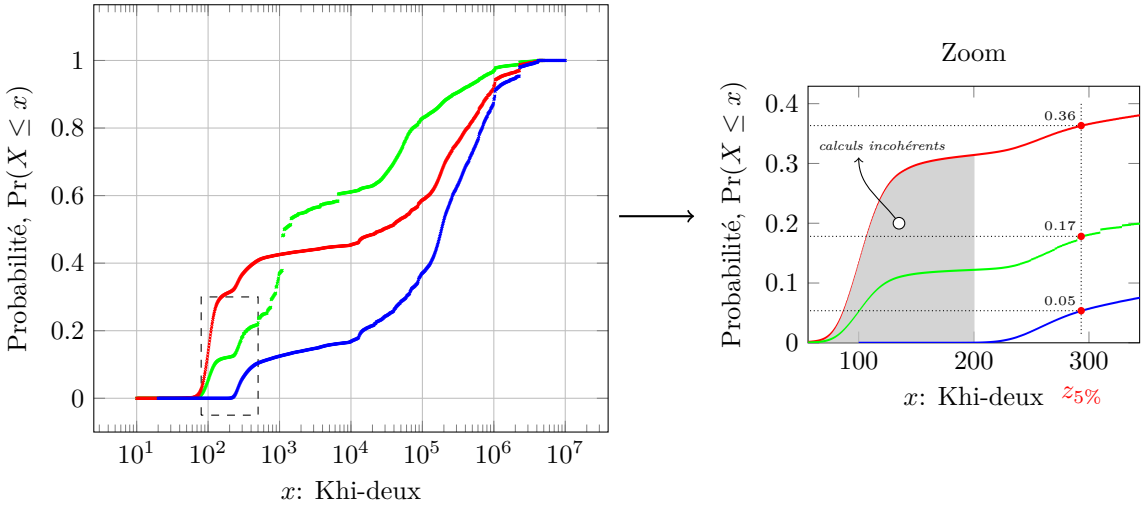

FIGURE 6.9 – Trois fonctions de répartition du test du Khi-deux pour 17 jours de capture sur une machine saine. (●) représente l'ensemble des requêtes, (●) les requêtes de taille au minimum 2Ko et (●) les requêtes de taille au minimum 2Ko en excluant les requêtes du processus *Kaspersky*.

La FIGURE 6.10 présente la distribution du test du rapport de vraisemblance, énoncé dans la partie 5.2.2, pour 17 jours de capture. Notre modèle se comporte bien par rapport au monde réel : 84% des requêtes en écriture lui sont favorable. Moins de 1% des statistiques des requêtes sont très aberrantes, c'est-à-dire supérieures à 160. Le processus *Kaspersky* ne pose pas de problème ici, seul 68 de ses requêtes concernent des en-têtes de fichiers.

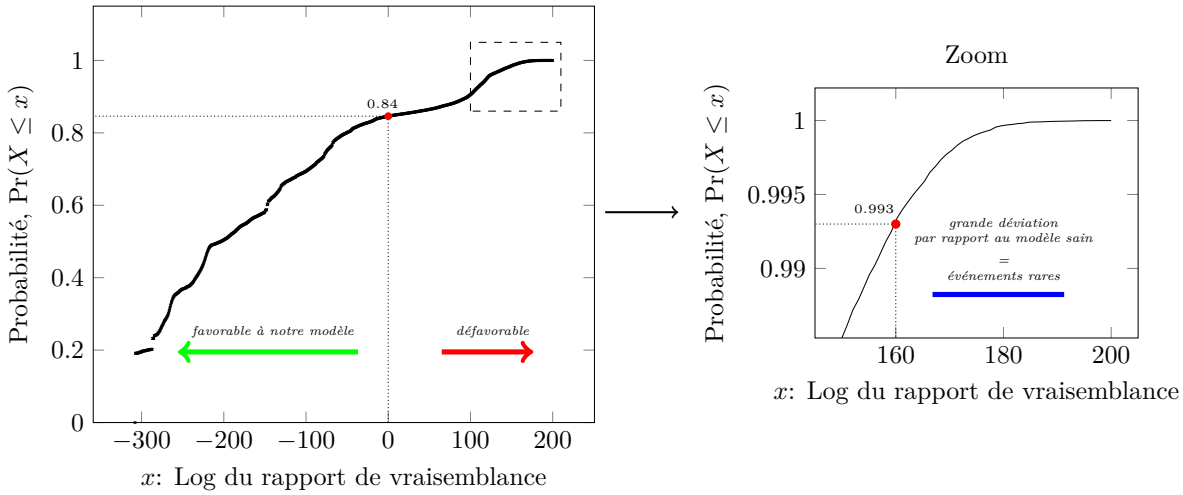

FIGURE 6.10 – La fonction de répartition du log du test du rapport de vraisemblance pour 17 jours de capture sur une machine saine.

Distribution des données : attaque de logiciels de rançon

Les FIGURES C.3 et C.4 permettent de visualiser la distribution des indicateurs de compromission pour chacun des 131 logiciels de rançon du corpus. Le corpus est visible annexe C et est composé d'échantillons récupérés en juin 2017. Celui-ci est un sous-ensemble de la campagne 3 annexe A, même si des différences sont notables pour les labels, ceux-ci ont été récupérés à des dates différentes. Les échantillons sont donc plus récents que ceux de la partie précédente.

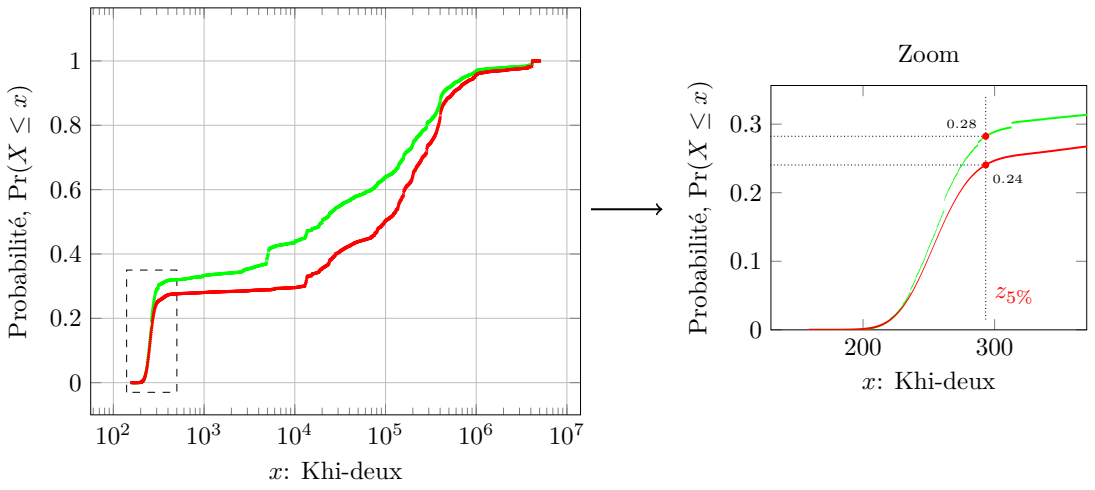

FIGURE 6.11 – Deux fonctions de répartition du test du Khi-deux pour les 131 logiciels de rançon du corpus. (●) représente l'ensemble des requêtes, (●) les requêtes de taille au minimum 2Ko.

La distribution des statistiques du Khi-deux, FIGURE 6.11, est bien différente de celle observée lors d'un usage courant. 24% des requêtes en écriture sont inférieures à la valeur de seuil $z_{5\%}$. La moitié des requêtes en écriture sont favorables à notre chaîne de Markov FIGURE 6.12. 14% des statistiques sont supérieures à 160 (i.e., grande déviation), c'est 1% dans un contexte courant.

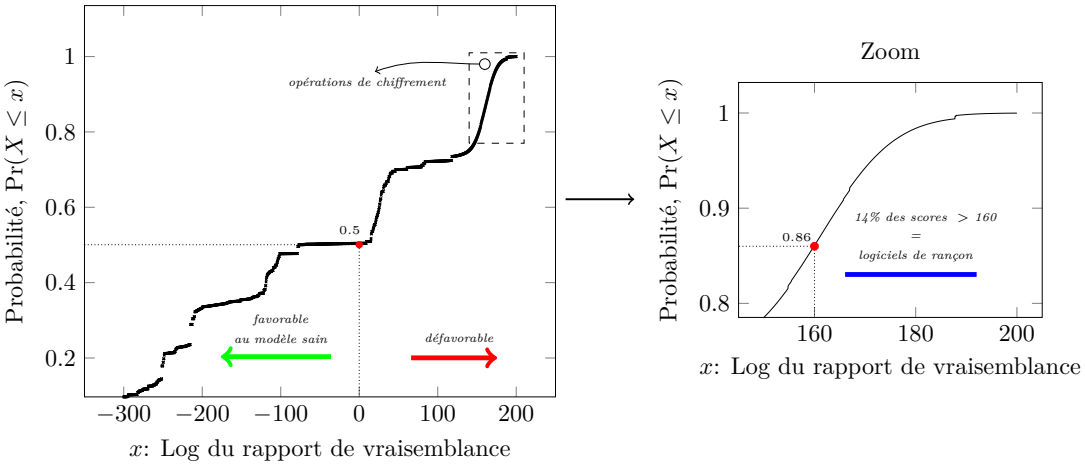

FIGURE 6.12 – La fonction de répartition du log du test du rapport de vraisemblance pour les 131 logiciels de rançon du corpus.

6.2.4 Mise au point des niveaux d'alertes

Nous choisissons de mettre au point deux niveaux d'alertes dans cette seconde version de DaD. L'objectif est de faire baisser le taux de faux positifs. Pour cela, des ressources plus coûteuses sont déployées une fois le premier niveau d'alerte déclenché. Il s'agit d'une preuve de concept perfectible qui peut évoluer au gré des utilisateurs et des besoins (e.g., contrôle plus fin).

De l'état courant au niveau 1

Nos deux indicateurs de compromission en l'état doivent être indépendant, c'est-à-dire se suffire à eux-même. Ils sont complémentaires afin de couvrir le plus grand nombre d'usages malveillants du système de fichiers. Le fonctionnement de DaD v2.0 est donc très simple. Le premier des deux indicateurs qui détecte un fil d'exécution malveillant procède à son blocage, et ce même si le second indicateur est toujours dans l'état courant pour ce fil. Nous fixons dans cette partie les valeurs critiques de nos deux indicateurs qui déclenchent le passage au niveau 1. Le passage de l'état courant, c'est-à-dire bénin, au niveau 1 est transparent pour l'utilisateur et cette surveillance est peu coûteuse pour le système (i.e., un seul type d'I/O request packet).

Test du rapport de vraisemblance Il s'agit ici de régler le taux de faux positifs à partir du jeu de données bénins. Le taux de vrais positifs est le meilleur possible d'après le lemme de Neyman-Pearson, définition 5.1, pour un taux de faux positifs choisi. La FIGURE 6.13 présente la probabilité d'occurrence d'une requête en fonction de la valeur de seuil du test. Petit rappel, nous disposons de 1.45 million requêtes d'écriture. Les faux positifs sont désignés par les requêtes en écriture incorrectement classifiées FIGURE 6.13 pour une valeur de seuil fixée. Par la suite nous utilisons également les fils d'exécution comme métrique pour exprimer le taux de faux positifs.

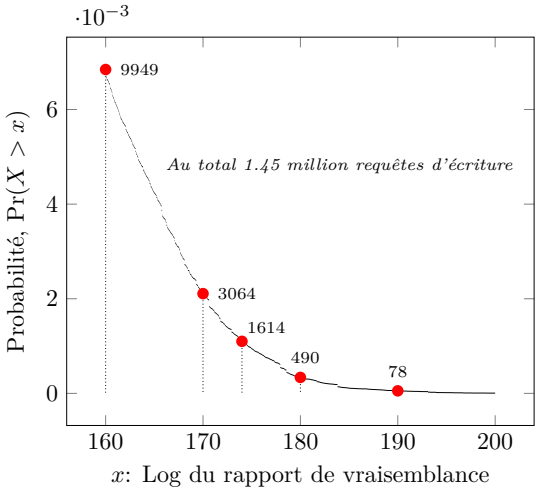

FIGURE 6.13 – La fonction de survie du log du test du rapport de vraisemblance pour 17 jours de capture sur une machine saine.

En choisissant par exemple un taux de faux positifs $q = 0.11\%$, nous obtenons 1614 faux positifs (i.e., requête en écriture) sur une période de 17 jours. Soit une alerte toutes les 8 minutes [9], ce qui reste bien trop élevé. Nous sommes alors tentés d'augmenter la valeur de seuil mais dans ce cas attention aux faux négatifs. En pratique, le nombre de fausses alertes seraient sans doute moindre car les requêtes d'écriture d'un fil d'exécution ne sont pas indépendantes, elles sont corrélées. Ainsi une fois le niveau 1 atteint, il est probable que le fil d'exécution suspect, engendre de nouvelles requêtes de même nature sans incidence sur le taux de faux positifs.

La FIGURE 6.14 montre 98.3% des formats de fichiers qui déclenchent des faux positifs dans la configuration ci-dessus. Les 1.7% restant sont disséminés entre les extensions : `.pset`, `.dbx`, `.vch`, `.tmp`, `.txt`, `.dat`, `.bin`. Les fichiers de cache du navigateur *Firefox* génèrent la moitié des faux positifs. Nous trouvons ensuite pour 23% des fichiers dont l'extension ne peut être récupérée. Nous ignorons pour quelle raison technique. Les fichiers `.manifest` [10] contiennent des métadonnées et sont utilisés pour installer ou configurer des applications. Enfin, les fichiers `.model` sont manipulés par le service de sauvegarde à Inria : *CrashPlan*.

9. si l'on considère les 212 heures de capture comme contiguë et non la plage de 17 jours
10. https://fileinfo.com/extension/manifest

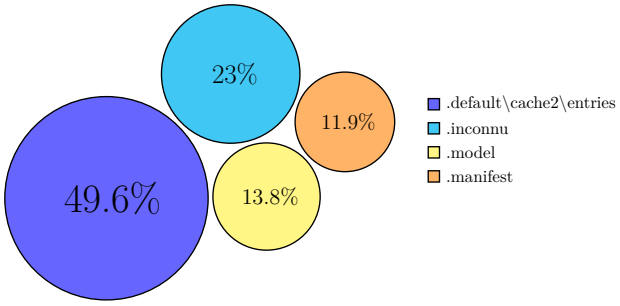

FIGURE 6.14 – Les types de fichiers qui déclenchent des faux positifs pour la valeur de seuil : 174.

La FIGURE 6.15 présente les applications à l'origine des faux positifs, toujours dans la même configuration que ci-dessus, c'est-à-dire $q = 0.11\%$. *Firefox* ainsi que le système sont les principales sources de faux positifs, c'est-à-dire 79% à eux deux. DaD surveille les requêtes d'écriture des fils d'exécution de l'espace utilisateur. Certaines de ces écritures sont déléguées au système via un mécanisme de cache (i.e., pagination [28]). C'est pourquoi le processus *System* est présent FIGURE 6.15. Certains logiciels de rançon exploitent ce mécanisme, il est donc impératif de surveiller ce type spécifique de requêtes. Le contenu de ces requêtes est plus hétérogène car elles peuvent provenir de différentes applications. Dans le cas présent, *Firefox* doit déléguer une bonne partie de ces I/O au système. Car nous savons que 49.6% des extensions qui déclenchent des faux positifs sont dues à *Firefox*, FIGURE 6.14. Les logiciels de sauvegarde *Crashplanservice* et *Dropbox* suscitent respectivement 7.6% et 5.6% de faux positifs. Le reste des applications utilisées par l'utilisateur se comportent bien par rapport à notre estimateur (i.e., déviation moins aberrante). Cet exemple nous montre qu'il est possible de réduire jusqu'à la moitié des faux positifs en ajoutant dans le corpus d'apprentissage de notre chaîne de Markov des fichiers de cache *Firefox*.

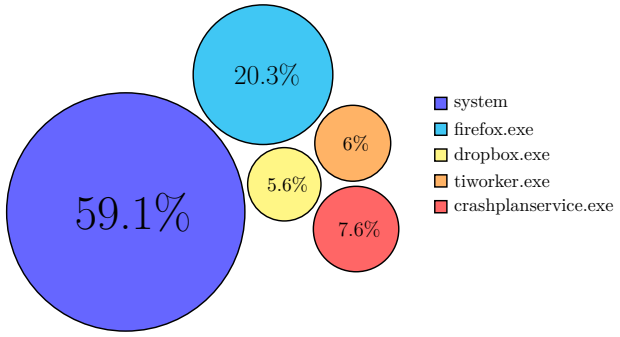

FIGURE 6.15 – Les applications qui déclenchent des faux positifs pour la valeur de seuil : 174.

Passage au niveau 1 Nous utilisons une heuristique très simple. Si au cours de la vie d'un fil d'exécution trois requêtes d'écriture ont une statistique du test du rapport de vraisemblance au dessus de la valeur critique $c = 180$, alors le fil d'exécution passe au niveau d'alerte 1. Dans cette configuration, l'utilisateur accepte de perdre dans le meilleur des cas 3 fichiers. En effet, des requêtes en écriture chiffrées peuvent avoir une statistique inférieure à 180. Nous prenons en compte l'historique complet d'un fil d'exécution et ce quelque soit le temps écoulé entre les requêtes. Nous allons voir dans le sous-paragraphe suivant que le fait de placer une contrainte temporelle ne change pas grand chose. L'avantage est que si un logiciel de rançon chiffre lentement les fichiers de l'utilisateur, celui-ci est quand même détecté. La valeur critique c a été déterminée de manière empirique. Les fils d'exécution malveillants ont été marqués manuellement dans les traces afin de s'assurer de l'exactitude des résultats.

Détermination de la valeur critique Les logiciels de rançon chiffrent très rapidement les fichiers FIGURE 6.16. Nous définissons Δ_t comme le temps maximum toléré entre 3 requêtes d'écriture qui dépassent la valeur critique c. Le taux de détection pour un Δ_t de $100\,ms$ et $10\,ms$ n'est pas trés éloigné du taux de détection sans contrainte temporelle pour $c \geq 150$.

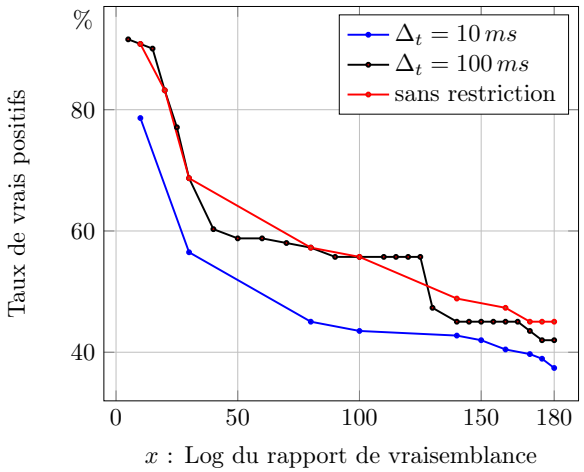

FIGURE 6.16 – Le taux de vrais positifs en fonction du log du test du rapport de vraisemblance.

Les résultats obtenus sont assez mitigés, pour un taux de faux positifs convenable, c'est-à-dire de 1.88% avec une valeur critique du test du rapport de vraisemblance fixée à 140, nous sommes capable de détecter seulement 45% des logiciels de rançon du corpus. En effet la famille *Cerber* ne touche pas aux en-têtes et représente plus ou moins 20.45% du corpus. Un autre phénomène plus embêtant est le fait que la famille *TeslaCrypt* modifie les en-têtes, mais le score obtenu ne dévie pas suffisamment. En fait, les échantillons *TeslaCrypt* sont détectés pour des statistiques trop faibles, c'est-à-dire comprises entre 20 et 40. Nous obtenons ainsi au mieux un taux de détection de 83% avec $c = 20$ sans contrainte temporelle, FIGURE 6.16. Les dix-sept pour-cent restant correspondent aux échantillons de la famille *Cerber*. Malheureusement une telle valeur critique c ne peut être utilisée, elle engendre trop de faux positifs, FIGURE 6.17. Les échantillons de la famille *TeslaCrypt* comptent pour 25% du corpus. Pour résumer des opérations très aberrantes

(e.g., chiffrement) sur les en-têtes ne sont réalisées que par 45% des échantillons. Par contre nous obtenons un taux de faux positifs extrêmement faible sur le jeu de données sains pour une valeur critique c correctement choisie.

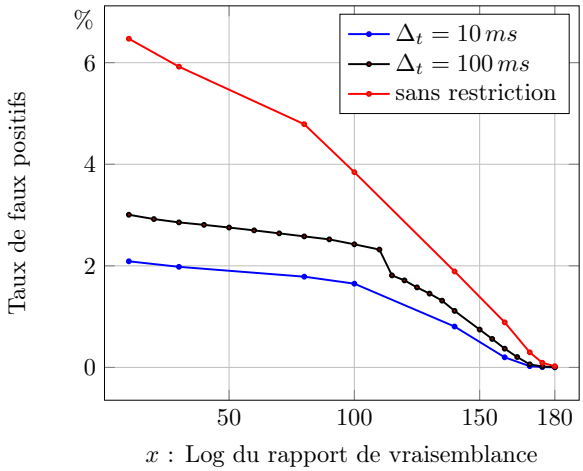

FIGURE 6.17 – Le taux de faux positifs en fonction du log du test du rapport de vraisemblance.

Nous choisissons la valeur critique $c = 180$ sans contrainte temporelle. Nous obtenons ainsi un taux de faux positifs de 0.023% pour un taux de détection de 45% des échantillons du corpus. 0.023% de faux positifs correspond à 24 fils d'exécution pour plus de 100k fils d'exécution dans les traces. Il est possible de baisser ce taux de faux positifs en ajoutant une contrainte temporelle. Dans les mêmes conditions avec un $\Delta_t = 100\,ms$ nous obtenons seulement 7 fils. Le coût du niveau 1 sous-section 6.2.4 est raisonnable, il est donc préférable de ne pas mettre de contrainte. Ainsi la solution est résiliente face aux logiciels de rançon qui temporisent leur attaque.

Coût pour l'utilisateur Dans la moitié des cas l'utilisateur perd au plus 2.3Mo avant de passer au niveau 1, FIGURE 6.18. Un peu plus de 70Mo sont perdus dans le pire des cas pour 90% des échantillons. L'utilisateur perd au plus 68 fichiers dans 50% des cas d'après la FIGURE 6.19. Il perd également au plus 454 fichiers pour 90% des échantillons, ce qui est beaucoup trop important. Les résultats ne semblent pas excellent mais en réalité ils sont sans doute meilleurs. En effet, l'ensemble des requêtes d'écriture d'un fil d'exécution malveillant sont considérées dans ces deux statistiques jusqu'au passage au niveau 1. Un grand nombre d'entre eux créent de nombreux fichiers avant et pendant le chiffrement (e.g., dépaquetage, notes de rançon, etc). Des données n'appartenant pas à l'utilisateur sont donc considérées comme perdues. Il faudrait traiter manuellement chaque fil d'exécution malveillant pour avoir des statistiques exactes.

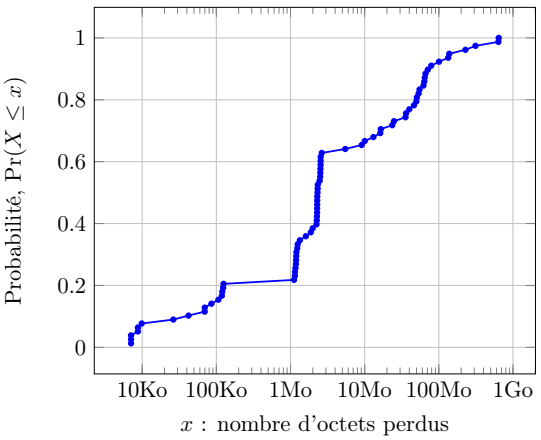

FIGURE 6.18 – La fonction de répartition des fils d'exécution malveillants, et pour chaque fil le nombre d'octets perdus.

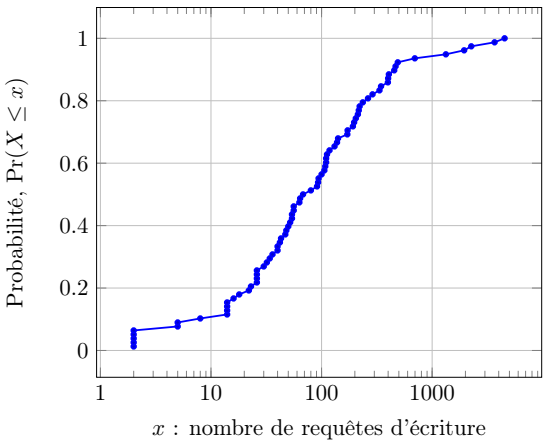

FIGURE 6.19 – La fonction de répartition des fils d'exécution malveillants, et pour chaque fil le nombre de requêtes d'écriture.

Test du Khi-deux Nous avons démontré précédemment que le test du Khi-deux est efficace pour détecter le chiffrement des données utilisateurs par un logiciel de rançon [PDB+17]. Mais de nombreux faux positifs sont inhérents à cette méthode de détection. L'utilisation des niveaux d'alerte 1 et 2 permet de limiter les faux positifs. Pour le passage au niveau 1 d'un fil d'exécution nous instaurons une contrainte temporelle Δ_t entre la première et la dernière opération contenue dans le vecteur utilisé pour calculer la médiane du Khi-deux.

Passage au niveau 1 Nous utilisons donc l'heuristique suivante :

— si médiane du Khi-deux sur les 50 dernières requêtes en écriture est inférieure à $z_{5\%}$,

— si $\Delta_t \leq 1$ seconde (i.e., $T_{op_{49}} - T_{op_0}$),

alors le fil d'exécution correspondant passe au niveau d'alerte 1. L'ajout d'une contrainte temporelle réduit le taux de faux positifs sans impacter le taux de vrais positifs, comme nous allons le voir dans le sous-paragraphe suivant. En effet, la grande majorité des logiciels de rançon du corpus chiffrent rapidement les données de l'utilisateur. Les applications légitimes qui chiffrent ou compressent des données sont sans doute peu nombreuses dans le jeu de données sain et manipulent moins de fichiers (i.e., comportement sporadique), ce qui étire dans le temps Δ_t.

Détermination de la fenêtre temporelle FIGURE 6.20, le taux de vrais positifs est identique que l'on utilise ou non une contrainte temporelle sur les 50 dernières requêtes en écriture pour peu que $\Delta_t \geq 1$ seconde. Le taux de vrais positifs est alors de 96.9%.

Les 4 échantillons non détectés appartiennent à trois familles différentes d'après *Avclass*, mais ils exhibent le même comportement. Leurs opérations de chiffrement possèdent des statistiques du Khi-deux trop élévés pour être détectées. Il y a deux explications possibles : (1) ajout de données avec une faible entropie ou (2) utilisation de primitives cryptographiques faibles. Nous passons de 1397 fils d'exécution suspects sans contrainte temporelle, FIGURE 6.21, à seulement

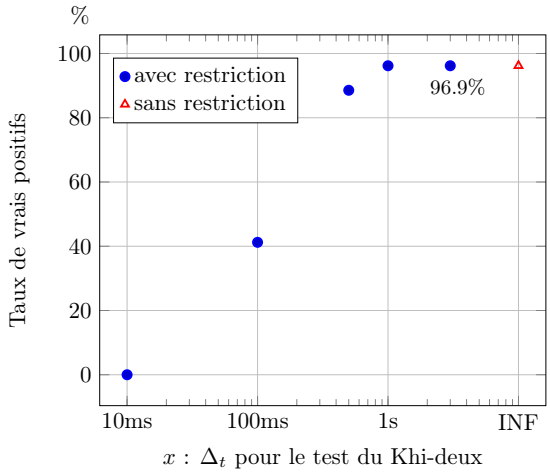

FIGURE 6.20 – Le taux de vrais positifs en fonction de la fenêtre temporelle Δ_t et du Khi-deux.

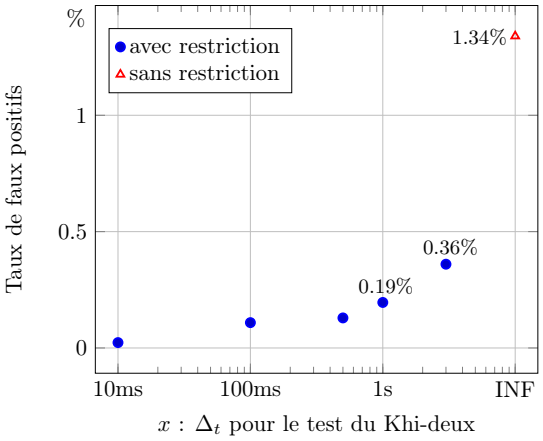

FIGURE 6.21 – Le taux de faux positifs en fonction de la fenêtre temporelle Δ_t et du Khi-deux.

203 avec un Δ_t de au plus 1 seconde. Pour rappel, il y a plus de 100k fils d'exécution dans les traces au total. Nous disposons donc d'un taux de détection de 96.9% des échantillons du corpus pour 0.19% de faux positifs parmi les fils d'exécution avec un Δ_t de au plus 1 seconde.

L'ajout d'une contrainte temporelle réduit significativement les faux positifs, c'est-à-dire de 85%. Un effet de bord se manifeste alors : un logiciel de rançon "plus lent" n'est pas détecté. De plus, la contrainte temporelle peut varier en fonction du système, de sa charge, des logiciels installés, etc. Il nous faut donc être prudent. Notre estimation de cette contrainte est empirique à partir des 17 jours de capture, et peut être propre à chaque système. Malgré tout, l'utilisation de $\Delta_t = 1$ seconde est relativement lâche pour que l'on puisse se soustraire des variabilités de performance de différents systèmes. Ce dernier point est à vérifier ultérieurement.

Coût pour l'utilisateur Nous pouvons voir FIGURE 6.22 que l'utilisateur perd au plus 100Mo et 15Mo dans respectivement 90% et 50% des cas. FIGURE 6.23 l'utilisateur perd au plus 300 fichiers dans 90% des cas. Les résultats médiocres sont sans doute meilleurs en réalité. En effet comme expliqué précédemment des opérations d'écritures sont prisent en comptes alors qu'elles ne concernent pas les données de l'utilisateur. Il s'agit de fichiers créés par les logiciels de rançon eux-même. Le test du Khi-deux est plus réactif que le test du rapport de vraisemblance si l'on considère le nombre de requêtes nécessaire avant le passage au niveau 1.

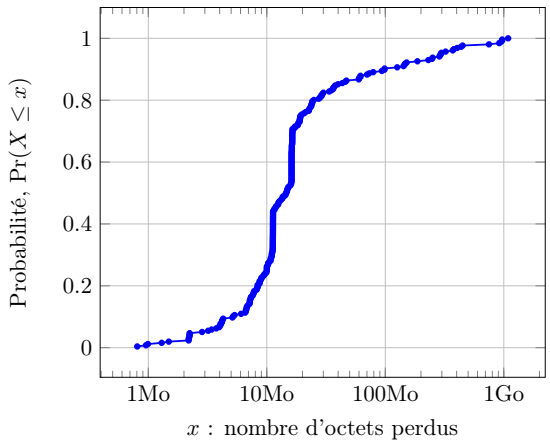

FIGURE 6.22 – La fonction de répartition des fils d'exécution malveillants, et pour chaque fil le nombre d'octets perdus.

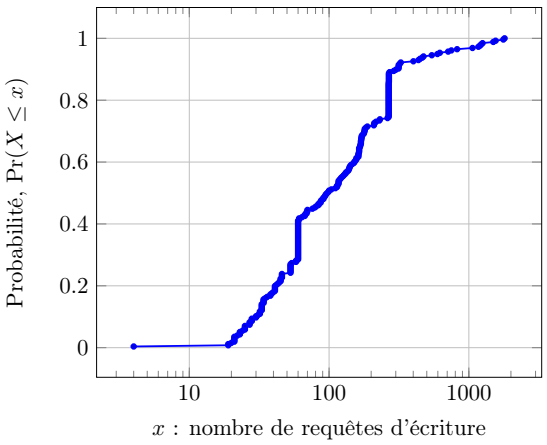

FIGURE 6.23 – La fonction de répartition des fils d'exécution malveillants, et pour chaque fil le nombre de requêtes d'écriture.

Du niveau 1 vers le niveau 2

L'utilisation conjointe des deux indicateurs ci-dessus nous permet de détecter 100% des logiciels de rançon au niveau 1. Ces deux indicateurs sont donc complémentaires. Une fois le niveau 1 atteint, les fichiers manipulés par le fil d'exécution suspect sont sauvegardés dans un espace sécurisé. DaD gère les accès à cet espace disque et est le seul à pouvoir y accèder en écriture. Pour pouvoir prendre une décision quant à la nature du fil d'exécution, il est nécessaire de le laisser s'exécuter pour observer son comportement. Ainsi, nous proposons de le laisser fonctionner 10 minutes avant de lancer un second test statistique. Le niveau 2 correspond à une infection et engage le blocage du fil d'exécution malveillant et la restauration des fichiers. Si la menace n'est pas avérée à l'issue du test présenté ci-dessous on redescend au niveau courant.

Afin de décider si une activité est malicieuse, nous proposons de comparer deux traces d'exécution capturées en temps réel. La première correspond à une trace saine ou tout du moins supposée saine, c'est-à-dire les statistiques des requêtes d'écriture des dix minutes précédant le passage d'un fil d'exécution au niveau 1. La seconde est constituée des statistiques de T_0, à savoir le moment ou un fil suspect passe au niveau 1, jusqu'à $T_0 + 10$ minutes. Pour chacune de ces deux traces, nous agrègeons les requêtes d'écriture de tous les fils d'exécution du système afin d'avoir une vision d'ensemble. Nous souhaitons ensuite déterminer si les deux jeux de données diffèrent de manière significative, pour cela nous utilisons le test de Kolmogorov-Smirnov [21]. Le test utilise alors les statistiques du Khi-deux ou du test du rapport de vraisemblance. Il s'agit d'un test non paramétrique (i.e., pas d'hypothèse sur la distribution des données) qui requiert uniquement une distributions continue des échantillons. De plus, la taille des échantillons peut être inégale. Le test de Kolmogorov-Smirnov calcule en fait la distance maximale entre deux fonctions de répartition empirique, équation (6.2). Cette distance notée D est la déviation verticale maximale entre les deux courbes, par exemple $S_{N_1}(x)$ et $S_{N_2}(x)$. Celle-ci n'est pas affectée par la nature de l'échelle utilisée, bien pratique pour le test du Khi-deux (i.e., logarithmique).

$$D = \max_{-\infty<x<\infty} \left| S_{N_1}(x) - S_{N_2}(x) \right| \tag{6.2}$$

$S_{N_1}(x)$ et $S_{N_2}(x)$ sont deux fonctions de répartition empirique

Nous passons d'une contre-mesure sans mémoire dans la contribution [PDB+17] à une contre-mesure qui conserve l'activité récente de tous les fils d'exécutions du système. Nous stockons uniquement les statistiques pour chaque requête et non les données. Chaque requête en écriture est liée à une structure `OPERATION` qui fait 60 octets, sans prendre en compte les questions d'alignement mémoire. L'empreinte mémoire de notre pilote est donc tout autre maintenant. En partant sur une estimation haute de l'activité d'un système, c'est-à-dire 100 processus qui possèdent 10 fils d'exécution et qui accèdent chacun en écriture sur une période de 10 minutes 10k fois au système de fichiers, alors DaD a besoin de 600 $Mo \times 2$. L'historique des opérations récentes peut-être stocké dans un fichier sur le disque, aucune contrainte ne s'applique sur le temps d'accès à ces données. En effet, la gestion des niveaux d'alerte dans DaD est asynchrone et non bloquant pour le système de fichiers. Nous pouvons également imaginer que ces opérations soient stockées sur un serveur local. L'espace nécessaire au stockage de ces informations n'est pas un problème, notamment en raison de l'utilisation toujours croissante d'espace disque et de mémoire vive sur les machines moderne. Nous pouvons également optimiser la structure de données `OPERATION`, ce qui n'est pas le cas pour le moment. Nous choisissons arbitrairement de conserver un historique de 10 minutes, mais celui-ci peut-être modulé. Nous savons que 90% des logiciels de rançon chiffrent les données en un maximum de 5 minutes [83].

Peu de temps après le démarrage de DaD si un fil d'exécution passe au niveau 1, nous ne disposons pas des 10 minutes précédentes pour le test de Kolmogorov-Smirnov. Deux solutions : (1) utiliser une trace de référence que nous incorporons dans le pilote pour adresser ce cas précis au démarrage ou (2) avant chaque arrêt on sauvegarde sur le disque les 10 dernières minutes qui seront utilisées au prochain redémarrage. Nous pouvons pour un gain de place sauvegarder la fonction de répartition empirique des dix dernières minutes et non toutes les statistiques.

Test du rapport de vraisemblance Nous souhaitons donc déterminer de manière empirique une distance D qui corresponde à une déviation suspecte entre des activités bénignes, c'est-à-dire les 10 minutes avant T_0, et des activités pour lesquelles nous nous posons des questions, à savoir les dix minutes suivantes. Pour cela, nous comparons avec le test de Kolmogorov-Smirnov :

1. chacune des traces malveillantes avec 1 trace saine unique et aléatoirement choisie,
2. deux traces saines contiguës dans le temps, et cela répété 100 fois.

Notre jeu de données dispose de plus de 1260 traces saines de 10 minutes. Nous obtenons ainsi deux fonctions de répartition empirique : $S_{N_1}(x)$ qui représente la distribution de la statistique D du test de Kolmogorov-Smirnov entre les traces malveillantes et saines et $S_{N_2}(x)$ qui représente la même chose mais pour des traces saines entre elles. Bien entendu nous nous limitons aux logiciels de rançon détectés par le test du rapport de vraisemblance soit 45% de la base étalon. Ce qui correspond à 59 échantillons sur 131. Le résultat est visible FIGURE 6.24.

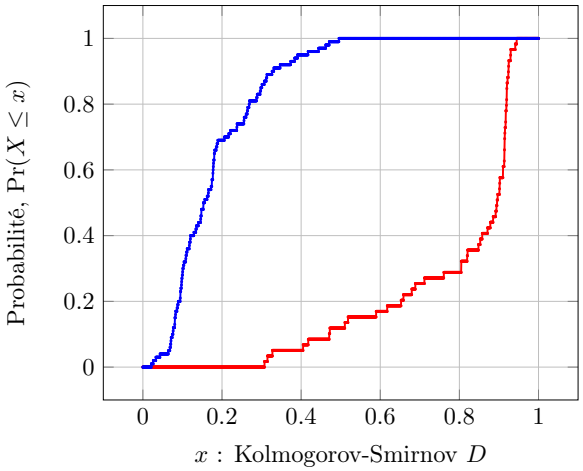

FIGURE 6.24 – Deux fonctions de répartition de la statistique D du test de Kolmogorov-Smirnov. (•) la distribution de la statistique D observée avec le test du rapport de vraisemblance entre des traces saines et (•) même chose mais entre des traces malveillantes et saines.

Nous observons une différence très nette entre les deux courbes. Ainsi pour un seuil de signification $\alpha = 0.05$, c'est à dire 5% de faux positifs dans le cas d'usage courant (•) nous obtenons la distance $D = 0.39$. Cette distance nous donne un taux de détection de 95% (•). Notre approche est probabiliste, les résultats peuvent donc varier. En utilisant la valeur critique déterminée ci-dessus, les simulations nous donnent dans le pire cas 3 échantillons non détectés. Finalement, les deux niveaux d'alerte du test du rapport de vraisemblance détecte seulement 2 fils d'exécution bénin comme malveillant sur 17 jours de capture. Au total 56 échantillons sur les 131 du corpus sont détectés. Le test de Kolmogorov-Smirnov est simple à mettre en œuvre et sa statistique D simple à interpréter. Dans le cas présent, nous avons comparé des traces saines obtenues sur un Windows 10 avec des traces malveillantes provenant de Windows 7. Malgré tout, les résultats sont bons. Pour plus de rigueur, effectuer les tests avec le même OS serait plus adéquat pour paramètrer la solution *offline*. La détection bien entendu est très dépendante de l'activité récente de l'utilisateur. La distance entre les deux fonctions de répartition FIGURE 6.24 est significative. 88% des statistiques D obtenues entre les traces malveillantes et saines (•) sont si extrêmes (i.e., supérieure à 0.47) que leur probabilité est nulle quand nous comparons des traces saines entre elles (•). Nous sommes donc capable de discriminer un logiciel de rançon à partir de l'activité récente du système de fichiers.

Test du Khi-deux Nous répètons les mêmes opérations que ci-dessus mais avec le test du Khi-deux. Les résultats sont plus mitigés, FIGURE 6.25. Nous obtenons un taux de détection de 90% (•) pour 8% de faux positifs (•) avec la statistique $D = 0.28$. Les simulations avec cette statistique détectent 116 échantillons sur les 131 du corpus et 17 fils d'exécution bénins sont marqués à tort comme malveillant. Le fait que des applications légitimes compressent, chiffrent, etc., explique les déviations moins importantes entre les traces malveillantes et saines et uniquement saines. En effet, 5% des requêtes en écriture sur la FIGURE 6.9 ont des statistiques inférieures à $z_{5\%}$.

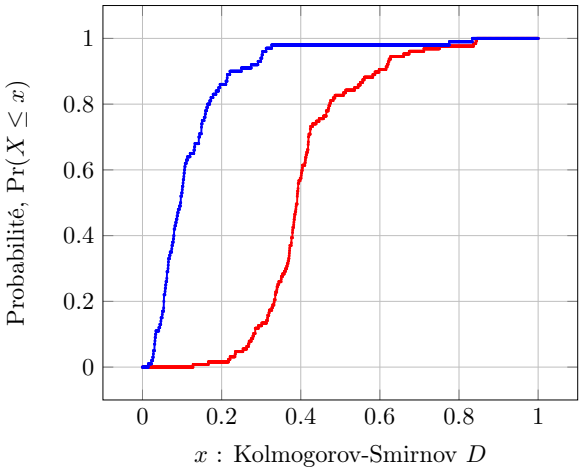

FIGURE 6.25 – Deux fonctions de répartition de la statistique D du test de Kolmogorov-Smirnov. (●) la distribution de la statistique D observée avec le test du Khi-deux entre des traces saines et (●) même chose mais entre des traces malveillantes et saines.

6.2.5 Synthèse

L'agrégation des deux indicateurs de compromission et des deux niveaux d'alerte permet de détecter 93.1% des échantillons. 9 logiciels de rançon ne sont pas détectés sur les 131 du corpus. Dans cette configuration le taux de faux positifs est de 0.019%, c'est-à-dire 19 fils d'exécution bénins incorrectement classifiés. L'utilisateur volontaire reçoit alors en moyenne un peu plus d'une alerte par jour avec les réglages et le jeux de données des paragraphes précédant. Ce qui reste convenable, mais encore perfectible. Nous pouvons imaginer l'ajout d'une boite de dialogue pour confirmer une attaque et ainsi éliminer les derniers faux positifs. La FIGURE 6.26 synthétise les résultats de nos expérimentations.

Les 9 faux négatifs sont issus de trois familles : *Teslacrypt*, *Cerber* et *Bitman*. Une analyse manuelle des traces révèle qu'ils n'utilisent pas de mécanisme particulier pour évader la détection. Les échantillons du corpus sont tous détectés par le premier niveau d'alerte si nous agrégeons les deux indicateurs de compromission. Ils ont donc un comportement "classique". Le test de Kolmogorov-Smirnov qui intervient en second rempart est inefficace dans certaines situations. La cause est simple : la trace bénigne utilisée comme "référence". Mais ce n'est pas la seule explication puisque qu'aucun échantillon de la famille *Locky*, c'est-à-dire la plus représentée dans le corpus, ne fait partie des faux négatifs. Le comportement de ces 9 échantillons ou de leurs familles n'est sans doute pas assez extrême en ce qui concerne la distribution des statistiques de nos indicateurs de compromission en comparaison de certaines traces saines.

Le premier niveau d'alerte est centré sur un fil d'exécution, le second s'intéresse au comportement global du système. Nous souhaitons ainsi déterminer si le comportement d'un fil d'exécution a une incidence trop importante sur la distribution globale des requêtes d'écriture du système de fichiers pour que celui-ci soit bénin. Si plusieurs fils d'exécution chiffrent en parallèle, la détection n'est que plus aisée. Nous comparons la distribution des requêtes d'écriture avant et après le passage d'un fil au niveau 1. Nous obtenons ainsi deux fonctions de répartition qui sont comparées

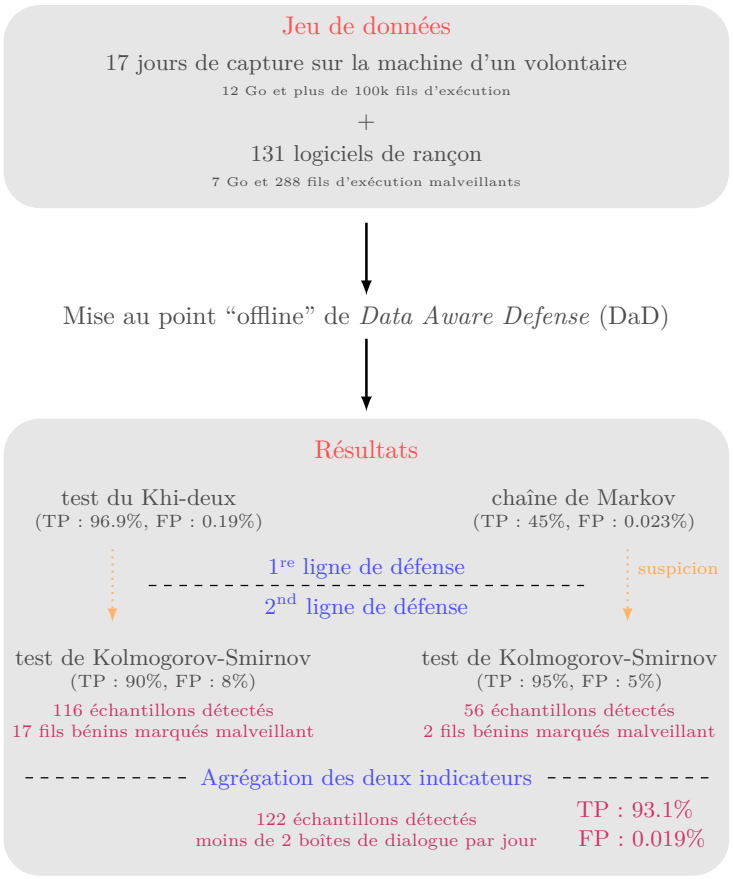

FIGURE 6.26 – Les résultats de nos expérimentations avec DaD.

avec le test de Kolmogorov-Smirnov. Si la distance entre les deux courbes est supérieure à une valeur critique alors une activité malveillante est détectée. La détection est très dépendante de la trace supposée "saine" utilisée lors de la comparaison. Mais le comportement agressif des logiciels de rançon est tel que les résultats sont satisfaisant.

Les statistiques des opérations sont enregistrées en continu par DaD et celui-ci conserve un historique pour les 10 dernières minutes. Le coût associé est invariant que un ou plusieurs fils soient au niveau 1. En revanche, le coût de la sauvegarde des fichiers touchés pour chaque fil d'exécution au niveau 1 s'additionne. Nous ne disposons pas d'une évaluation des performances de DaD v2.0 car il est difficile de rendre compte du surcoût pendant qu'un ou plusieurs fils d'exécution sont au niveau 1, c'est-à-dire 227 pour 17 jours de capture. Au final, rien ne change concernant la routine qui intercepte les requêtes en écriture. Seule la routine asynchrone qui calcule les statistiques d'une opération effectue deux traitements supplémentaires : (1) calcul du test du rapport de vraisemblance et (2) ajout des statistiques de l'opération dans le fichier journal du fil d'exécution correspondant. Ces deux nouveaux points ne sont pas bloquant pour le système de fichiers. De plus, l'utilisateur volontaire n'a pas ressenti de baisse des performances avec la version "passive" de DaD. En dessous d'un certain seuil d'utilisation de la mémoire vive nous pouvons garder les statistiques des fils d'exécution en mémoire (i.e., ce qui est le cas pour

la collecte des traces sur MoM [11] et sur la machine de l'utilisateur volontaire). Nous ajoutons également un ramasse miette afin de ne conserver que l'historique voulu des opérations sur le système de fichiers. Nous pouvons imaginer allouer une quantité de mémoire vive fixe pour stocker les X dernières statistiques de chaque fil. Sur les versions 64-bit de Windows, la quantité de mémoire virtuelle adressable dans le noyau est considérable. En effet, celle-ci n'est limitée que par la quantité de mémoire volatile ou presque [27] (i.e., jusqu'à 8 To).

Les résultats obtenus dans cette section sont valables sur notre jeu de données. Des expérimentations supplémentaires avec de nouveaux logiciels de rançon mais aussi davantage d'utilisateurs sont à prévoir pour vérifier le comportement de DaD. Le paramètrage de DaD peut-être ajusté en fonction des besoins, par exemple favoriser un faible taux de faux positifs au détriment des capacités de détection et inversement. Nous pouvons également imaginer que notre solution se calibre automatiquement en fonction de la machine utilisateur. Que ce soit pour notre chaîne de Markov, mais aussi pour ajuster la distance D du test de Kolmogorov-Smirnov sur les traces saines de l'utilisateur en question. La taille de l'espace sécurisé qui sauvegarde les fichiers touchés par un fil d'exécution au niveau 1 n'est pas déterminée. Cet espace peut-être victime d'un déni de service. De nombreuses options peuvent être ajoutées : partage des traces entre machines locales, remontée régulière des traces vers un Security Information and Event Management System, etc.

Cette seconde version de DaD nécessite encore quelque investigations concernant ses paramètres, il s'agit du processus d'industrialisation. Celle-ci n'est pas disponible, car elle utilise le procédé que nous avons breveté. De plus, la création d'une start-up et d'un produit de sécurité sont des options envisagées. Nous prévoyons de déployer cette version de DaD sur la machine de l'utilisateur volontaire mais cette fois en activant la protection temps réel. L'utilisateur pourra alors nous notifier si des baisses de performance se font sentir.

6.3 Conclusion

La première partie du chapitre, section 6.1, propose une contre-mesure simple mais efficace, qui est la seule à véritablement adresser le problème des performances lors de sa présentation en novembre 2017 [12]. Nous regrettons qu'une comparaison plus détaillée de celle-ci ne soit possible avec les autres contributions du domaine. La conception minimaliste de DaD possède les caractéristiques suivantes :

1. contre-mesure générique,
2. détection comportementale,
3. une seule IRP (i.e., écriture),
4. un seul indicateur de compromission (i.e., Khi-deux),
5. un faible coût sur le système.

La première itération de DaD souffre d'une limitation majeure : le nombre de faux positifs. De part sa conception celle-ci détecte en fait des requêtes en écriture ayant une forte entropie. Pour

11. nous avons expérimenté un manque de mémoire avec 4Go sur Windows 7 32-bit pour certains échantillons
12. Kharraz *et al.* [88] présentent indépendamment Redemption

rappel, nous n'utilisons pas de liste blanche, tous les fils d'exécution de l'espace utilisateur sont pris en considération. En effet, Kim *et al.* [90] montrent que certains logiciels malveillants sont signés par une autorité de confiance et peuvent ainsi évader la détection des antivirus.

La seconde version, section 6.2, comble ce problème grâce à l'ajout d'un indicateur de compromission et de deux niveaux d'alerte. Cet indicateur n'est autre qu'une chaîne de Markov qui modélise les en-têtes de fichiers courants manipulés par un système sain. Les résultats sont visibles FIGURE 6.26, et ceux-ci sont satisfaisants. Des vérifications supplémentaires de ces résultats sont à prévoir par des techniques d'échantillonnages (i.e., cross-validation) sur de nouveaux jeu de données. Malheureusement nous ne présentons pas de comparaison de DaD avec des antivirus traditionnels ou des solutions dédiées aux logiciels de rançon. La raison principale est le manque de temps. Les performances de cette seconde version sont également à investiguer, même si les traitements supplémentaires quand aucun fil d'exécution n'est suspect ont un impact quasi nul. Un coût non négligeable est présent [13] quand un fil d'exécution est suspect, car les fichiers touchés sont alors sauvegardés en attendant que le second niveau d'alerte se prononce 10 minutes plus tard. Le niveau d'alerte 1 est déclenché à tort par 227 fils d'exécution pendant les 17 jours de capture, soit 13 fois par jour en moyenne. Le niveau d'alerte 2 quant à lui est déclenché à tort par moins de 2 fils d'exécution par jour en moyenne. Cette seconde version de DaD laisse s'exécuter un fil d'exécution au comportement suspect, l'objectif est de collecter davantage de "preuves" quant à la nature de celui-ci. La sauvegarde des fichiers touchés pendant cette intermède est donc primordiale. Les FIGURES 6.9 et 6.10 montrent que nos deux indicateurs se comportent bien dans le monde réel et que ceux-ci ont du sens. Le fait que le niveau d'alerte 1 se déclenche souvent avec un coût associé non nul, nous encourage à mettre en place un troisième indicateur de compromission. Celui-ci ne doit plus s'intéresser à la nature des requêtes en écriture mais à des métriques concernant l'utilisation du système de fichiers. En effet, des attaques plus sophistiquées (e.g., mimétisme) peuvent leurrer nos deux indicateurs, un tel indicateur ne serait alors pas dupé. Le chapitre 7 propose des pistes que nous avons commencé à explorer dans ce sens, mais qui devront être poursuivis dans des travaux futurs.

13. le coût est peut-être imperceptible pour l'utilisateur

Chapitre 7

Travaux en cours et futurs

Ce chapitre présente des travaux en cours concernant le comportement des utilisateurs dans un contexte de cybersécurité ainsi que des pistes pour des travaux futurs relatives au choix d'un indicateur de compromission supplémentaire.La première partie du chapitre présente des résultats qui sont le fruit d'une collaboration avec le laboratoire d'observation des usages des TIC de l'Université de Rennes 2. Les deux parties suivantes présentent des travaux préliminaires concernant un troisième indicateur de compromission. Les discussions dans ce chapitre sont donc moins formelles et fournies que précédemment.

7.1 Le dilemme des utilisateurs face à un programme suspect

Notre contre-mesure DaD est susceptible de générer un nombre important de faux positifs, dans ce contexte un message d'alerte ou d'avertissement peut-être présenté à l'utilisateur afin de réduire leur nombre. La prise de décision lui est alors déléguée. Nous avons donc cherché à mettre en place une expérience utilisateur à Inria afin de tester le comportement des utilisateurs lors de l'affichage d'une boîte de dialogue sur leur écran. Des expériences sur des utilisateurs nécessitent de passer un comité d'éthique, en raison des implications juridiques et psychologiques. Nous avons donc mis en place une collaboration appelée *WARNING* avec un laboratoire spécialiste des interactions homme-machine : le LOUSTIC [1] à l'Université de Rennes 2. Celui-ci possède les outils, la compétence et les autorisations nécessaires pour réaliser des expériences sans avertir préalablement l'utilisateur du sujet d'étude. Dans notre cas, une simulation d'attaque d'un logiciel de rançon, celle-ci pouvant entraîner un stress pour le participant. Un protocole expérimental strict est donc requis.

7.1.1 Le contexte théorique

Nous ne cherchons pas à présenter un état de l'art exhaustif des messages d'alerte dans le contexte de la cybersécurité, mais plutôt à présenter brièvement les points les plus importants. Pour plus d'informations, vous référez au travail de Patrick Dreyfus [74] du LOUSTIC. Le premier

1. http://www.loustic.net/

point et pas des moindres est qu'il n'existe aucune norme de conception pour l'utilisation de message d'alerte en cybersécurité. Les raisons qui poussent un utilisateur à faire confiance à tel ou tel programme ne sont pas claires. Bien que les risques de l'Internet soient connus ou relayés en interne par les entreprises, ceux-ci sont souvent négligés par les utilisateurs. De même, les utilisateurs ne conçoivent pas qu'ils puissent eux même être une victime. Des signes de confiances factices peuvent aussi être utilisés par les criminels afin de berner la confiance de l'utilisateur.

La perception du risque joue un rôle important dans la prise de décision, celle-ci est variée en fonction des utilisateurs. Nous pouvons citer différentes raisons à cela comme des habitudes, croyances, opinions ou une confiance excessive dans un système informatique.

Nous souhaitons via le message d'alerte communiquer avec l'utilisateur à propos d'un danger. Au premier abord nous sommes dans le flou sur la façon de procéder pour évaluer le comportement des utilisateurs mais aussi concernant les facteurs à prendre en compte. Tout en gardant en tête que seule une quantité limitée de boîtes de dialogues peuvent être testées. Il s'agit donc d'orienter l'utilisateur vers la bonne décision, et ce, quelque soit le contexte dans lequel celui-ci utilise sa machine. Patrick Dreyfus [74] identifie ainsi des facteurs psychologiques pour les messages d'alerte et des normes de conception adaptées au contexte de sécurité sur l'Internet. Nous en citons quelques uns :

— la perception du risque,

— l'habituation,

— le rapport des coûts et des bénéfices,

— l'influence sociale,

— le contexte,

— le niveau d'expertise.

De multiples facteurs peuvent donc influencer un utilisateur. Il est possible de les évaluer grâce à un questionnaire psychologique que les participants remplissent après un test. Des exemples de normes de conception sont :

— la couleur,

— le texte,

— les symboles,

— la compréhension,

— l'attention.

Le manque de clarté des messages d'avertissement est une des raisons qui poussent les utilisateurs à les désactiver. Les messages doivent être explicites et attirer l'attention. L'ergonomie du message joue aussi un rôle important, afin de ne pas nuire à l'information que nous souhaitons transmettre. La section suivante présente les résultats d'une expérimentation utilisateur menée avec le LOUSTIC en 2018.

7.1.2 Expérimentations et résultats préliminaires

Un panel de 51 participants entre 18 et 25 ans ont participé à l'expérimentation, ceux-ci viennent de tout domaine. Ils sont recrutés via la base de testeur du LOUSTIC et rémunérés avec des cartes cadeaux. Des informations sont récupérées avant chaque passation : âge, sexe, étude, système d'exploitation utilisé, etc. 17 participants viennent d'un cursus psychologie. 40 se déclarent expert en technologie, c'est-à-dire au niveau 5 et 6 du document Stages of Adoption of Technology Survey Instrument [37]. Sans grande surprise 42 participants ont pour habitude d'utiliser un système d'exploitation *Windows*. Les passations et le rapport de l'étude [75], sur lequel je m'appuie, ont été réalisés par Echraf Mekacher ingénieure d'études ergonome au LOUSTIC.

Expérimentations Les passations se sont déroulées du 15 au 24 janvier 2018. Celles-ci mobilisent du personnel du LOUSTIC, notamment afin de vérifier le comportement des utilisateurs lors des tests. Chaque passation comprend en fait 2 expériences distinctes. Une expérience principale (i.e., qui n'est pas le sujet ici) demande à l'utilisateur de réaliser plusieurs tâches sur une machine du laboratoire. Afin de ne pas introduire de biais, les participants sont découpés en quatre groupes pour les besoins des deux expériences. Bien entendu les participants ignorent qu'ils vont être victime d'un logiciel malveillant. Les expérimentations sont réalisées sans consentement préalable, le LOUSTIC possède les autorisations pour conduire ce type d'expérience. À la fin de la seconde tâche (i.e., 10 minutes environ) un message d'alerte fait son apparition en plein centre de l'écran. Ce message demande à l'utilisateur de prendre une décision, car un processus suspect accède à un document, celui-ci est donné avec son chemin absolu. Il s'agit en fait du document dans lequel le participant sauvegarde la tâche. L'utilisateur risque donc de perdre le travail qu'il vient d'accomplir. Le participant peut vérifier le contenu du document en naviguant dans le système de fichiers s'il le souhaite. La boîte de dialogue offre trois possibilités à l'utilisateur : (1) autoriser, (2) bloquer ou (3) quitter. Les trois actions conduisent au même résultat : la fermeture de la boîte de dialogue. Le participant n'est ensuite plus inquiété. Le message d'alerte est bloquant, c'est-à-dire que le participant ne peut pas réaliser la tâche suivante tant qu'il ne donne pas une réponse. La boîte de dialogue peut être de deux types, "danger" ou "attention", celles-ci sont présentées FIGURES 7.1 et 7.2.

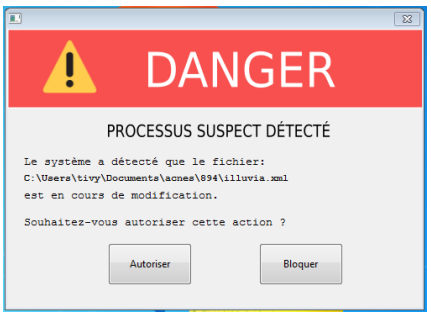

FIGURE 7.1 – Fenêtre "danger".

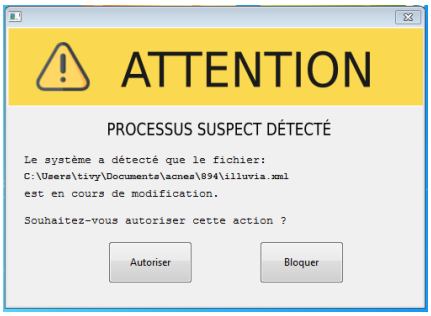

FIGURE 7.2 – Fenêtre "attention".

Seule la couleur et le texte du message principal de la fenêtre change. Une fois que le participant a effectué son choix, un membre du LOUSTIC vient dans la salle de passation pour évaluer

son ressenti vis-à-vis du message d'alerte. Pour cela, un questionnaire mesurant les affects positifs et négatifs entre 0 et 5 est utilisé, il s'agit du document Positive and Negative Affect Schedule (PANAS) [34]. Chaque passation est individuelle et le participant est seul dans une pièce.

Cette première expérimentation utilisateur a pour objectif de vérifier une hypothèse modeste : la boîte de dialogue de couleur rouge doit normalement entraîner plus de stress chez les participants que celle de couleur jaune. Les groupes "danger" devraient donc se montrer plus prudent et bloquer en plus grand nombre le processus suspect que les groupes "attention". Les effectifs dans les TABLEAUX 7.1 et 7.2 comprennent 43 participants et non 51. Lors des premières passations, un bug de l'interface de test est apparu empêchant la collecte des résultats.

Résultats Le TABLEAU 7.1 nous montre qu'une majorité des participants, au nombre de 20, n'ont pas souhaité prendre une décision claire, ils ont simplement fermé la fenêtre. Ils justifient ce choix en indiquant qu'il ne s'agit pas de leur ordinateur et que ce n'est donc pas à eux de prendre cette décision. Un facteur psychologique rentre également en ligne de compte : l'habituation à ce type de message intempestif les conduit par réflexe à fermer la fenêtre. Néanmoins 7 participants du groupe "attention" et 10 du groupe "danger" bloquent la modification du fichier. Ils avancent deux raisons pour motiver ce choix : (1) selon eux c'est le choix évident et (2) ils font habituellement la même chose sur leur ordinateur. Les actions ne sont pas influencées par le type de la fenêtre, "attention" ou "danger". En effet, un test du Khi-deux bilatéral ne rejette pas l'hypothèse nulle qui est : il n'y a pas de différence significative entre les deux effectifs. La statistique du Khi-deux est 0.474 pour 2 degrés de liberté, soit une p-value de 0.789.

Tableau 7.1 – (Mekacher *et al.* [75]) Les décisions prises par les participants lors de l'affichage du message d'alerte, selon le type d'interface.

	Type boîte de dialogue	
	Jaune - Attention	Rouge - Danger
Autoriser	3	3
Quitter	9	11
Bloquer	7	10

Les résultats du TABLEAU 7.2 sont hétérogènes. Les participants technophiles, niveaux 5 et 6, ne sont que 11 à bloquer le processus suspect. 16 ferment la fenêtre et 6 autorisent même la modification du fichier. Ces utilisateurs semblent habitués à de tels messages, du moins ils ne perçoivent pas de risque. Aucun des utilisateurs non-technophile n'autorise l'action par contre.

2. Stages of Adoption of Technology Survey Instrument [37]

Tableau 7.2 – (Mekacher *et al.* [75]) Tableau croisé entre les décisions prises lors de l'affichage de la fenêtre d'alerte et le niveau de technophilie.

		Niveau de technophilie [2]				
		1	3	4	5	6
Décision	Autoriser	0	0	0	1	5
	Quitter	0	2	2	5	11
	Bloquer	1	3	2	5	6

Le TABLEAU 7.3 présente la moyenne des affects ressentis par les participants avec le questionnaire PANAS [34]. Les affects positifs sont quasiment identiques entre les deux groupes. Une différence notable est présente concernant les affects négatifs. Les participants du groupe "danger" ont un ressenti en moyenne presque deux fois plus négatif que ceux du groupe "attention". Un test de Student [108] est utilisé pour vérifier si les moyennes des deux groupes pour les affects négatifs diffèrent de manière significative. Il est probable d'après le seuil de significativité (i.e., $t(39.7_{ddl}) = 3.14,\ p\text{-}value = 0.003$), que les valeurs observées proviennent de deux populations différentes, c'est-à-dire que les différences ne sont pas dues au hasard.

Tableau 7.3 – (Mekacher *et al.* [75]) Moyenne des affects ressentis selon le groupe.

	Positif	Négatif
Groupe "Danger"	1.175	1.265
Groupe "Attention"	1.085	0.69

7.1.3 Synthèse

Finalement, nous constatons que les participants du groupe "danger" ressentent davantage d'affects négatifs que ceux du groupe "attention". La différence est significative avec un test de Student. Le point le plus troublant est qu'aucune différence significative ne se manifeste entre ces deux groupes concernant la prise de décision lors de l'affichage du message d'alerte. Les utilisateurs sont trop habitués à ce genre de message intempestif et n'y prêtent plus guère attention. Leur réponse est donc fonction de leur opinion. Le message d'alerte cible le fichier qui contient les résultats de la dernière tâche. Celle-ci dure 10 minutes, pour augmenter l'enjeu pour le participant, une durée plus importante est à considérer dans les travaux futurs.

Kharraz *et al.* [88] sont les premiers à réaliser des expériences avec des utilisateurs. Leur objectif est d'analyser le comportement des utilisateurs face à leur message d'alerte dans différents contextes d'utilisation. Ils ne testent qu'une seul boîte de dialogue qui offre deux possibilitées à l'utilisateur : autoriser ou refuser. Les choix de conception de celle-ci ne sont pas discutés. Ils souhaitent s'assurer qu'un utilisateur grâce aux indications de leur contre-mesure, REDEMPTION, soit capable de distinguer un faux positif d'une attaque. Ils réalisent pour cela un test de Student avec un effectif de 28 personnes qui suggère avec une p-value de 4.9×10^{-7} que REDEMPTION aide les utilisateurs à prendre la bonne décision. Plus de détails dans l'état de l'art sous-section 2.2.2.

Dans des travaux futurs, en plus de continuer d'investiguer l'ergonomie du message d'alerte, une étude du comportement des utilisateurs dans différents contextes est également nécessaire. En effet, l'utilisateur doit être capable de distinguer des opérations suspectes mais légitimes dont il est à l'origine, d'actions suspectes intentées contre son gré ou sans son aval. Le travail de fin d'étude de Patrick Dreyfus [74] du LOUSTIC s'inscrit dans ce prolongement. Il propose des préconisations concernant la conception du message d'alerte, son étude est très complète. Je n'en discute que brièvement, car je n'y ai pas participé. Les résultats de son travail sont basés sur de nouvelles expériences utilisateurs, et pour cela il utilise notamment le suivi de l'œil pour voir ou se porte l'attention des participants.

7.2 Détection d'événements rares

Nous proposons d'utiliser un indicateur de compromission supplémentaire pour réduire encore davantage le taux de faux positifs de notre contre-mesure. Nous investiguons pour cela, l'utilisation d'une métrique simple, qui prenne en compte l'activité globale du système. Trois métriques sont proposées sur les FIGURES 7.3, 7.4 et 7.5. Nous comparons ainsi les résultats obtenus avec un échantillon d'un logiciel de rançon et une trace d'un système sain. Des différences significatives sont visuellement perceptibles. La trace du système sain provient de la machine de l'utilisateur volontaire. Une étude comparative plus complète est nécessaire pour choisir laquelle de ces trois métriques est la plus pertinente. Nous disposons de données, 131 traces malveillantes et des captures bénignes, mais il faut les analyser.

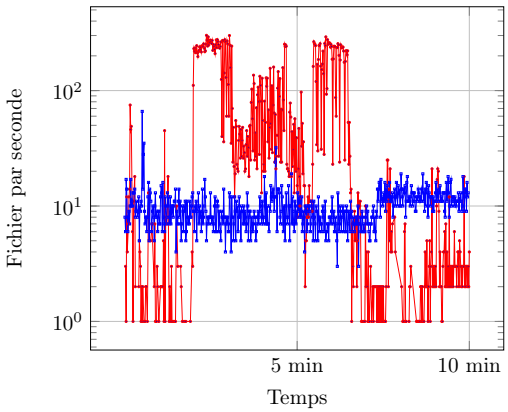

FIGURE 7.3 – Nombre de fichiers distincts ouvert en écriture par seconde : système infecté (●) et système sain (□).

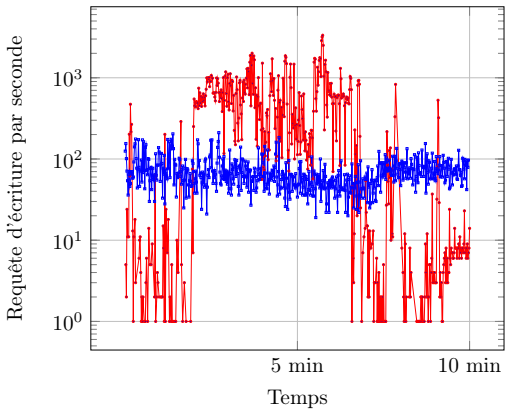

FIGURE 7.4 – Nombre de requêtes d'écriture par seconde : système infecté (●) et système sain (□).

Une fois la métrique choisie, nous proposons de baser notre détection sur la théorie des valeurs extrêmes [77]. Les statistiques de la trace saine choisie ci-dessus semblent avoir une distribution stationnaire, au moins pendant les 10 minutes de capture. Cette théorie permet de modéliser la distribution des queues de distribution et ainsi de détecter des valeurs très aberrantes. Son utilisation est très répandue dans de nombreux domaines. Nous souhaitons donc l'utiliser en

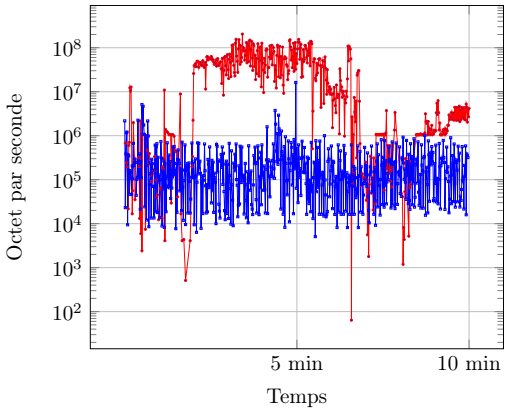

FIGURE 7.5 – Nombre d'octets écrit par seconde : système infecté (●) et système sain (□).

temps réel sur un flux de données (e.g., nombre d'octets écrit par seconde) pour détecter des anomalies. La contribution de Siffer [106] nous a mis sur la piste. Celui-ci l'utilise dans un IDS pour détecter des paquets malveillants sur le réseau. L'utilisation de cette théorie des valeurs extrêmes sur notre jeu de données est à considérer je pense.

7.3 Notes de rançon et collisions

Nous proposons également l'utilisation d'un autre indicateur de compromission basé sur la reconnaissance des "couches" du Khi-deux. Nous avons présenté cette particularité section 5.2.1 et FIGURE 5.5. Celle-ci est propre à certains logiciels de rançon. La reconnaissance de ces motifs permettrait de limiter les faux positifs. Il est dans l'intérêt du logiciel de rançon d'aviser l'utilisateur qu'il doit payer pour récupérer ses données, et pour cela, de nombreuses notes de rançon sont nécessaires. Nous avons réalisé une expérimentation simple qui consiste à calculer un haché sur le contenu de chaque requête en écriture. Deux requêtes en écriture identiques provoque donc une collision (e.g., métadonnées, notes de rançons, etc). La FIGURE 7.6 illustre cela pour des applications bénignes et des échantillons de logiciels de rançon.

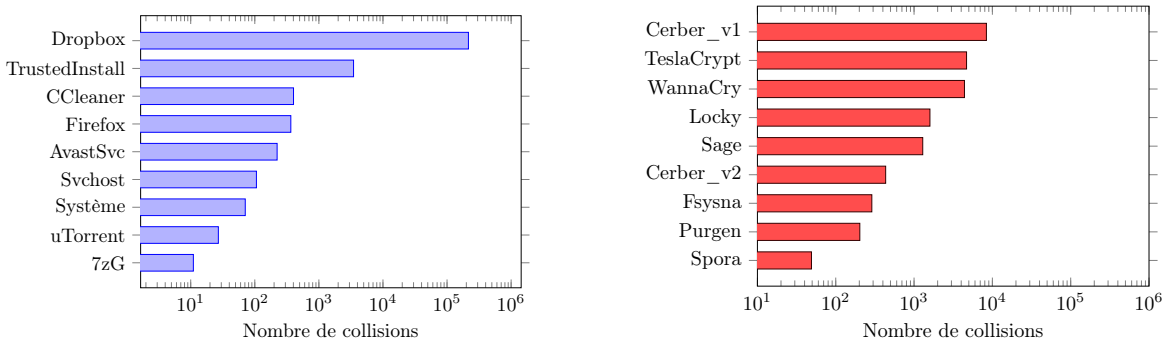

FIGURE 7.6 – Nombre de collisions sur les requêtes en écriture : logiciels bénins ■ et logiciels de rançon ■.

La proportion de collisions pour les logiciels de rançon est en moyenne plus importante, si l'on exclu *Dropbox*. Il s'agit du nombre de collisions pour 10k requêtes en écriture lors d'une attaque. Les statistiques choisies pour les applications bénignes correspondent aux plus extrêmes que j'ai trouvé en parcourant manuellement les résultats pour quelques-unes des captures (i.e., 10 minutes chacune). La plupart du temps, les résultats sont bien moins élevés. De plus, la distribution des statistiques du Khi-deux sur les données manipulées par un logiciel de rançon sont multimodales FIGURE 5.5. Il est probable que la distribution de cette même statistique soit unimodale pour les applications bénignes. Nous pensons donc qu'une heuristique permettant de reconnaître les "couches" est prometteuse pour limiter les faux positifs.

7.4 Conclusion

Nous avons vu dans ce chapitre qu'il reste encore de nombreuses pistes à explorer et approfondir. Notamment concernant la mise au point de notre contre-mesure, DaD. De nouvelles expérimentations sont nécessaires pour évaluer ses capacités de détection, mais aussi pour atteindre un taux de faux positifs aussi faible que possible.

Un peu à la marge du thème général de cette thèse, nous avons réalisé avec un laboratoire de sciences humaines, le LOUSTIC, des expérimentations en rapport avec le comportement des utilisateurs dans un contexte de cybersécurité ; soit les logiciels malveillants. Les conclusions de l'expérimentation présentée section 7.1 peuvent paraître minimes, mais ne reflètent en aucun cas la quantité de travail en amont nécessaire pour arriver à mettre sur pied un tel projet. Je terminerais en disant que la course à l'armement avec les logiciels de rançon ne fait que commencer et que nous sommes donc condamné à expérimenter sans cesse : de nouveaux indicateurs de compromission, messages d'alerte, etc.

Conclusion

La thèse traite de l'analyse et de la détection des logiciels de rançon, qui du début en 2015 de celle-ci jusqu'à sa fin en 2018 ont beaucoup fait parler d'eux. Dans le chapitre 1, nous introduisons les notions nécessaires à la compréhension de la thèse. Le chapitre 2 présente l'état de l'art, celui-ci est particulier car les contributions dans le domaine ne sont que très récentes pour la plupart et donc concurrentes à la thèse. Nous nous intéressons en particulier au comportement des logiciels de rançon, dans le but de déployer une contre-mesure additionnelle ou complémentaire à un antivirus classique. Deux approches sont proposées : (A) surveillance des accès aux librairies cryptographiques et (B) surveillance des I/O à destination des systèmes de fichiers. La seconde approche est retenue car plus générique et donc plus efficace. Les contributions dans la littérature disposent de taux de détection très élevés, quand celui-ci n'est pas de 100%. Les points faibles que nous avons identifiés sont : (1) les performances et (2) les faux positifs. En effet, il est compliqué de simuler le comportement courant d'un utilisateur et encore plus de disposer d'un protocole expérimental représentatif et reproductible. De plus, les contre-mesures les plus efficaces s'orientent vers une surveillance fine du système de fichiers et les performances s'en ressentent. Les effets de bords (1) et (2) rendent les contre-mesures inutilisables dans des cas d'usages réels.

Contributions

Avant même de commencer l'analyse de logiciels de rançon, il est nécessaire de disposer d'une plateforme d'analyse automatique de logiciels malveillants. Nous présentons donc Malware - O - Matic (MoM) chapitre 3. La plateforme est constituée de machines nues, et se démarque donc des contributions du domaine qui utilisent des machines virtuelles. L'objectif, bien entendu, est d'être le plus transparent possible pour inciter un binaire malveillant à déclencher sa charge utile.

Le chapitre 4 propose une contre-mesure temps réel dans l'espace utilisateur. Celle-ci implémente un fournisseur de service cryptographique (CSP) et espère ainsi intercepter tous les appels à la CryptoAPI. Son efficacité n'est pas suffisante et des biais dans notre preuve de concept nous empêche d'évaluer correctement cette approche. Nous abandonnons donc l'idée de concevoir une contre-mesure basée sur la CryptoAPI. Nous procédons ensuite de manière incrémentale dans les chapitres 5 et 6 pour adresser les effets de bords (1) et (2).

Data Aware Defense ; version 1

Nous proposons, chapitre 5, une contre-mesure temps réel agnostique qui s'intéresse au comportement des applications sur les systèmes de fichiers, en particulier les requêtes en écriture. Pour cela, nous implémentons un pilote de filtre dans l'espace noyau. Conscient de l'impact sur le système d'utiliser plusieurs indicateurs de compromission pour détecter un comportement malveillant, nous n'utilisons qu'un seul indicateur : rapide, robuste et simple à interpréter. Cet indicateur n'est autre que le test du Khi-deux, largement employé dans la littérature pour évaluer la qualité des générateurs de nombres aléatoires. Ici, son objectif est donc de détecter des opérations de chiffrement sur le système de fichiers. Cette approche minimaliste donne de bons résultats avec un taux de détection de 99.37%, mais est très limitée en terme de faux positifs. En effet, de nombreuses applications bénignes déclenchent de fausses alertes. Mais l'objectif de cette contribution n'est pas là, il s'agit d'adresser l'effet de bord (1). En effet, grâce à l'utilisation d'outils standards pour évaluer les performances, nous démontrons que notre contre-mesure, DaD, n'impacte que très légèrement le système de fichiers et plus généralement le système. De plus, nos expérimentations sont reproductibles, DaD est disponible et les hachés de nos logiciels de rançon le sont aussi. Deux observations lors des expérimentations sont à noter : (I) comportement en "couches" observé avec le test du Khi-deux sur les requêtes d'écriture, caractéristique des fils d'exécution malveillants et (II) deux faux négatifs cherchent à dissimuler leur comportement malveillant en mimant un comportement légitime. Dans cette configuration l'utilisateur perd au plus 69 Mo dans 90% des cas avant que DaD n'intervienne. Il nous reste encore à adresser le problème des faux positifs, nous le faisons chapitre 6.

Data Aware Defense ; version 2

Nous proposons l'ajout d'un second indicateur de compromission, en s'intéressant toujours à un seul type d'I/O c'est-à-dire les requêtes en écriture. Cet indicateur est une chaîne de Markov qui modèlise les en-têtes de fichiers courants manipulés par un système non infecté. À partir de notre modèle "sain" nous établissons ensuite la vraisemblance des requêtes sur les en-têtes. Nous étendons également DaD avec la présence de deux niveaux d'alerte. L'objectif est d'être plus réactif pour limiter les fichiers compromis mais aussi de sauvegarder les fichiers touchés en cas de doute entre les niveaux 1 et 2. Afin de mettre au point cette seconde version de DaD, celle-ci est déployée sur la machine d'un utilisateur pendant 17 jours mais aussi sur MoM. Le paramètrage est ensuite réalisé hors-ligne à partir des traces collectées. Le premier niveau d'alerte utilise le test du Khi-deux et le test du rapport de vraisemblance indépendamment. Ce premier niveau détecte avec la granularité d'un fil d'exécution les opérations suspectes. Le second niveau compare la distribution d'une des deux statistiques précédentes avant et après le passage au niveau 1 pour tout le système sur une fenêtre de dix minutes avec un test de Kolmogorov-Smirnov. Nous souhaitons observer un comportement suspect et ses effets sur la distribution globale des statistiques avant de prendre une décision, pendant ce temps les fichiers touchés par le fil d'exécution sont sauvegardés. Nous obtenons les résultats suivants : un taux de détection de 93.1% et un taux de faux positifs de 0.019%. Tous les logiciels de rançon sont détectés niveau 1 et l'utilisation conjointe des deux indicateurs a du sens, car ceux-ci sont complémentaires. Nous

observons néanmoins que certaines familles de logiciels de rançon ne touchent pas aux en-têtes et d'autres n'écrivent pas de données suffisamment déviantes. Pour être plus explicite, 9 échantillons sur les 131 ne sont pas détectés et 19 fils d'exécution bénins sont détectés comme malveillant à tord sur plus de 100k fils.

Nous n'avons pas la prétention de présenter cette seconde itération de DaD comme une solution optimale. Les points suivants doivent être travaillés : (i) il est nécessaire d'évaluer précisément l'impact de la solution une fois le niveau 1 atteint, en effet, la sauvegarde des fichiers est alors active, (ii) comparaison de DaD par rapport aux antivirus et solutions dédiées logiciels de rançon et (iii) réduire davantage encore les faux positifs, nous sommes en moyenne à moins de 2 fausses alertes par jour. Nous proposons malgré cela, une contre-mesure qui peut être déployée dans des cas d'usages réels, car nous adressons le problème des performances et des faux positifs.

Perspectives

Pour confirmer l'utilisabilité de DaD, son déploiement est à prévoir. En ce sens, nous avons mené des expérimentations sur des utilisateurs avec un laboratoire partenaire : le LOUSTIC.
Le chapitre 7 présente des résultats préliminaires concernant le comportement des utilisateurs face à un message d'alerte. L'objectif est la mise au point d'une boîte de dialogue, la plus pertinente possible. Par ailleurs, dans le chapitre 7 nous discutons également de l'ajout d'un troisième indicateur de compromission pour améliorer détection et précision. La dernière version de Data Aware Defense (DaD) n'est pas disponible, car il existe actuellement un projet de création de start-up à Inria dont l'objectif est son industrialisation. Des travaux menés par des doctorants Rennais sont également en cours concernant les logiciels de rançon au sujet de : (a) les traces ou artefacts réseaux et (b) les mécanismes d'évasion.

Récemment deux contributions ont proposé de nouvelles approches. Huang *et al.* [84] en 2017 présentent un SSD dont le firmware est modifié afin d'être résilient face à une attaque de logiciel de rançon. En juin 2018, Taylor *et al.* [109] présentent sur Linux un système de fichiers capable d'assurer la protection et l'intégrité des données. Il nous semble néanmoins plus facile et moins contraignant d'intégrer un pilote de filtre sur un poste client plutôt que modifier le firmware d'un matériel ou d'intégrer du code défensif dans un système de fichiers existant.

Nous évoquons aussi brièvement les dernières contributions de l'année 2018 concurrentes à la rédaction de ce manuscrit. En mai, Kharraz *et al.* [86] discutent des précédentes contre-mesures et des points à adresser. Ils identifient ainsi : (1) la détection des modules cryptographiques de l'attaquant, (2) précision de la détection, soit le problème des faux positifs et (3) la restauration des fichiers grâce à une contre-mesure qui ne requiert que peu de modifications de la machine client. Il est troublant que ceux-ci n'évoquent pas les performances. Mehnaz *et al.* [97] en septembre présentent une nouvelle contre-mesure temps réel : *RWGuard*. Celle-ci est la première à utiliser des fichiers de leurre et se positionne uniquement dans l'espace utilisateur. La détection d'un comportement malveillant est basé sur de l'apprentissage automatique et la surveillance du système de fichiers via l'utilisation de l'outil *IRPLogger*. Le taux de détection obtenu est de 100% et le taux de faux positifs de 0.1%. Mais les auteurs n'investiguent pas davantage les faux positifs (e.g., nombre de requêtes) et ne déploient pas leur contre-mesure sur la machine d'un

utilisateur. Les mesures de performances sont non reproductibles et la contre-mesure n'est pas disponible. Enfin, en novembre Genç *et al.* [80] présentent à la conférence *NordSec* des primitives cryptographiques que les logiciels de rançon pourraient utiliser pour évader la détection.

Au sujet de la menace, celle-ci peut viser de nouvelles cibles, tels que les voitures, les objets connectés, le matériel médical, la domotique, etc. Cette thèse traite uniquement des logiciels de rançon sur les ordinateurs personnels, puisqu'il s'agit d'une menace avérée et non hypothétique au moment où j'écris ces mots.

Annexe A

Corpus logiciels de rançon

campagne 1 du 16/02/2017

teslacrypt	127
xorist	125
cerber	102
bitman	69
zerber	24
yakes	23
deshacop	19
gpcode	13
locky	12
gamarue	9
fsysna	8
shade	7
usteal	5
troldesh	5
dalexis	5
crypmod	4
cryptowall	3
onion	3
crypmodadv	2
myxah	2
crysis	2
mikey	2
shifu	2
bkclient	1
SINGLETON:4ac1d572ce059b68ecb6eeb49a301448	1
SINGLETON:6d83d702fad47bf24a04c4b3e2c9d930	1
SINGLETON:decd769bd3b242f1d5538610634ec94c	1
SINGLETON:b7b6eae61c9bf2db1764aceb6ee1208a	1
zeqbeq	1
vbkryjetor	1
dycler	1
SINGLETON:128847bf71a3fa9c3b564c41cb541dc5	1
waldek	1
torrentlocker	1
SINGLETON:5da40d9a01b65625a8729623ab965117	1
teerac	1
SINGLETON:47967d7eb8d8789c45216bb535a00234	1
SINGLETON:137f2dcee6f3fd645a1c523b19a2d30d	1

SINGLETON:9b8c8a75a0aa189767d562d603516054	1
midie	1
fury	1
ranserkd	1
SINGLETON:76fc0ef0a9081773ccc12a1337f47ec4	1
SINGLETON:731350a08dfe3b1f194d7da54a2d8dab	1
purga	1
SINGLETON:b1efeb888b13378829057569d3ef1ed3	1
firecrypt	1
SINGLETON:b61754f00d2059cc7c79505525e4f993	1
farfli	1
SINGLETON:a202773eb36b26001ec46cbc0b9f2144	1
SINGLETON:e17dd24fb359ca4071fd7fc95eb9edc8	1
SINGLETON:046c31b39dfd7efa5529d967d9da0cd2	1
critroni	1
SINGLETON:c558f21781b5dd424e7a00d6e69a27aa	1
swisyn	1
SINGLETON:dd5cffa78a245a1409d3d049dd980431	1
virut	1
SINGLETON:f5873f8cbe463155e1df2544483bc801	1
genkryptik	1
mayarchive	1
SINGLETON:bc4a269ef127d108659149b6058ac7d8	1
zusy	1
rsaist	1
filecryptor	1
xmkfxc2cdrjc	1
SINGLETON:ce18c2fb763667050b645e33877f0acd	1
SINGLETON:0d330e378e1257206be148e514875449	1
nemucod	1
sality	1
lethic	1
SINGLETON:01e4c37d61c092a9c63170c2e4d73441	1
SINGLETON:5b499d2825b174f1d87caddb2aa64d50	1
SINGLETON:62da799689f1fa028572e529a36982a3	1
SINGLETON:7263864b3ff85cdf8f4f58ba64a5ba31	1
tescrypt	1
scatter	1
SINGLETON:797e48f7d26567ae870198f285539b9f	1

Sum: 627

campagne 2 du 11/04/2017

teslacrypt	68
cerber	33
bitman	32
shifu	7
locky	5
spora	3
zerber	3
yakes	2
zusy	2
genkryptik	2
ebowla	1
myxah	1
reconyc	1
torrentlocker	1
jigsaw	1
mikey	1
beebone	1
tescrypt	1
cryptxxx	1
cryptolocker	1
razy	1
SINGLETON:ec40d848d4bef75360a79d2b21d18a72	1
SINGLETON:a5ff68ec809d6d4d88a0714eab3f3588	1
enestedel	1

Sum: 171

campagne 3 du 02/06/2017

teslacrypt	80
cerber	57
bitman	18
zerber	15
shade	12
locky	7
confidence	7
troldesh	6
sagecrypt	5
shifu	5
spora	4
deshacop	3
yakes	3
purgen	3
wannacry	2
ranserkd	2
tescrypt	2
midie	1
nemucod	1
ishtar	1
vbkryjetor	1
scar	1
filecryptor	1
SINGLETON:ef84b1eab0a61dedcb7a449a2d457a74	1
SINGLETON:ce0eca422ed4f87fd0a8c742ec7f8bff	1
beebone	1
fsysna	1
zusy	1
cryptolocker	1
myxah	1
SINGLETON:a17fef2f0954699fbb1c62808762e282	1
crypmod	1
SINGLETON:18fa4bd7dd8d45ee147698f4088d4f02	1

Sum: 247

Les hachés des échantillons sont disponibles à l'adresse ci-dessous :
http://people.rennes.inria.fr/Aurelien.Palisse/corpus.html

Annexe B

Corpus chaîne de Markov

Corpus d'apprentissage

.gls	7
.sys	1
.jpg	14940
.dwf	18
.docx	37
.xml	1682
.ps	3441
.xlsx	10
.gif	3821
.ppt	11236
.pps	430
.fits	75
.sgml	16
.log	783
.text	282
.rtf	232
.123	1
.eps	321
.kml	28
.bmp	16
.wp	96
.swf	74
.doc	15047
.pub	6
.js	1
.dbase3	380
.kmz	45
.unk	860
.xls	7370
.tex	94
.png	603
.sql	71
.hlp	81
.troff	36
.csv	2106
.f	225
.tmp	33
.txt	24734

```
.pptx                79
.html                53122
.java                29
.pdf                 52289
.fm                  6
.gz                  2511
.odp                 1
==================
# extensions    : 45
# fichiers      : 197276
```

Corpus de test

```
.gls                15
.sys                3
.jpg                31267
.dwf                68
.docx               45
.xml                10025
.ps                 5286
.zip                3
.wk1                3
.wk3                1
.gif                6243
.ppt                10827
.pps                482
.bin                1
.fits               8
.ttf                8
.sgml               1
.log                5392
.text               79
.rtf                182
.fm                 6
.eps                2254
.kml                190
.bmp                7
.wp                 55
.txt                7267
.doc                13934
.pub                34
.dbase3             832
.kmz                144
.unk                2495
.xls                24201
.exported           1
.tex                2
.lnk                2
.sql                211
.hlp                316
.troff              10
.csv                3867
.ileaf              1
.f                  27
.png                1328
.tmp                81
.xbm                4
.swf                1112
.pptx               34
.html               30182
.java               69
.pdf                35707
.xlsx               7
.gz                 2985
.odp                1
==================
# extensions    : 52
# fichiers      : 197305
```

Annexe C

DaD v2.0

Corpus mise au point DaD v2.0 du 02/06/2017

locky	40
teslacrypt	33
cerber	22
bitman	11
zerber	4
scatter	3
sagecrypt	2
confidence	2
sage	1
globeimposter	1
myxah	1
deshacop	1
ranserkd	1
beebone	1
yakes	1
shifu	1
emotet	1
gryphon	1
mediamagnet	1
crypmod	1
shade	1

Sum: 131

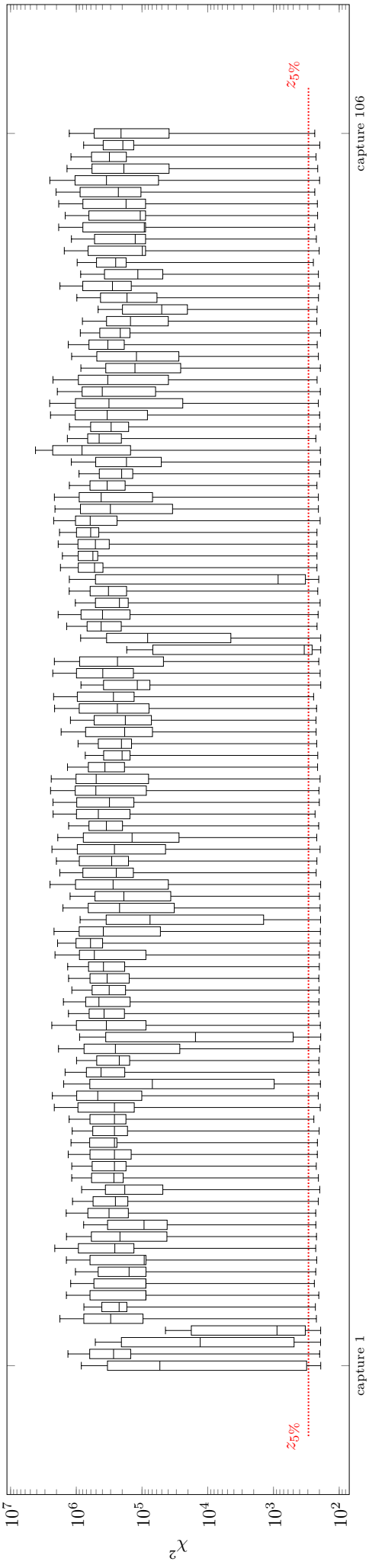

FIGURE C.1 – Évolution en fonction du temps de la statistique du Khi-deux sur une machine saine pour 17 jours de capture.

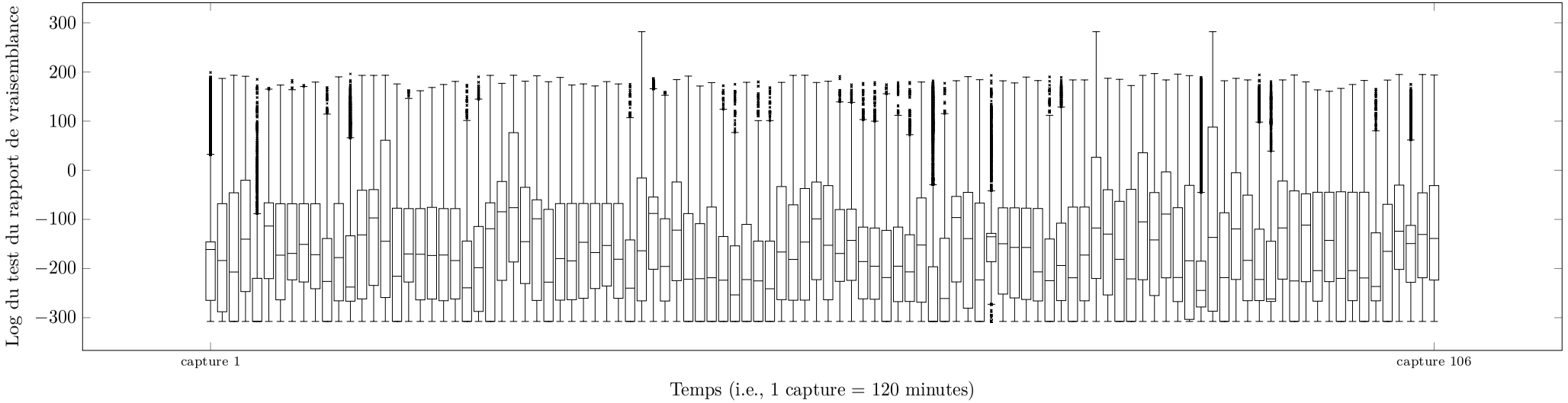

FIGURE C.2 – Évolution en fonction du temps du logarithme du test du rapport de vraisemblance sur une machine saine pour 17 jours de capture.

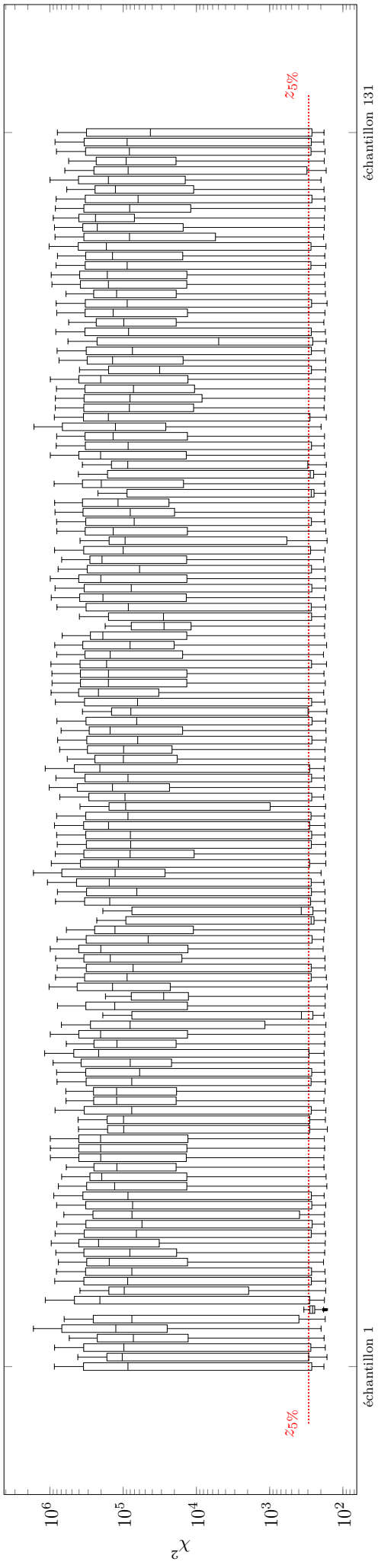

FIGURE C.3 – Distribution de la statistique du Khi-deux lors d'une attaque d'un logiciel de rançon.

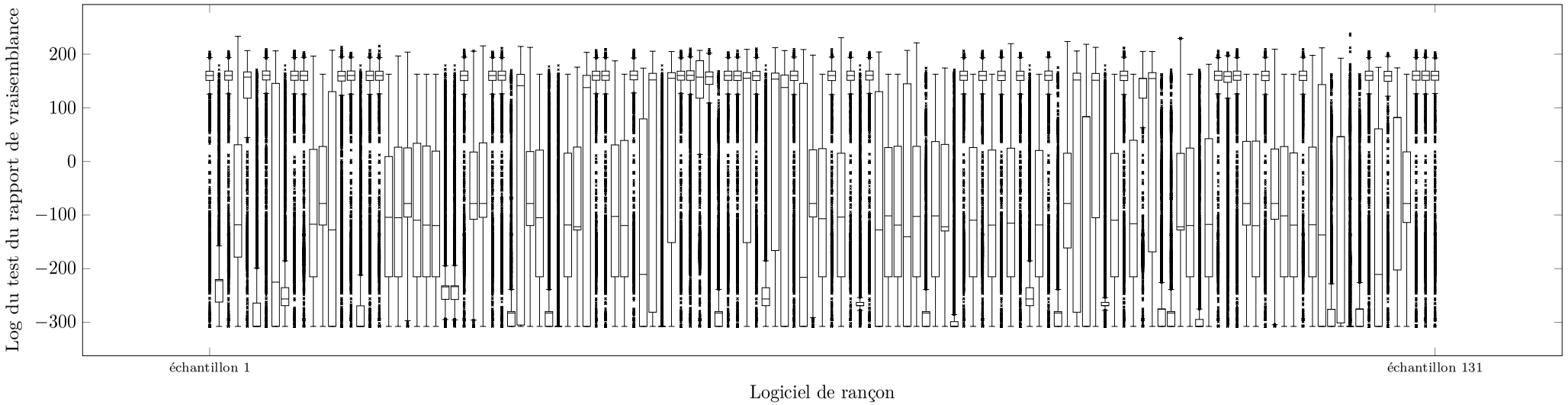

FIGURE C.4 – Distribution du logarithme du test du rapport de vraisemblance lors d'une attaque d'un logiciel de rançon.

Glossaire

AES	*Advanced encryption standard.*
ANSSI	*Agence Nationale de la Sécurité des Systèmes d'Information.*
CNG	*Interface de programmation d'applications cryptographiques de Microsoft qui remplace la CryptoAPI.*
Crypto++	*Une librairie d'algorithmes cryptographiques en C++.*
CryptoAPI	*Interface de programmation d'applications cryptographiques de Microsoft.*
DaD	*Data Aware Defense.*
GPT	*GUID partition table.*
HAL	*Hardware abstraction layer.*
IAT	*Import address table.*
IRP	*I/O request packet.*
JSON	*JavaScript object notation.*
LCN	*Logical cluster number.*
LOUSTIC	*Laboratoire d'observation des usages des TIC.*
MBR	*Master boot record.*
MFT	*Master file table.*
MoM	*Malware - O - Matic.*
NIST	*National Institute of Standards and Technology .*
NTFS	*New technology file system.*
PE	*Portable executable.*
PGP	*Logiciel de chiffrement à clé publique qui signifie : pretty good privacy.*

PRNG	*Générateur de nombres pseudo-aléatoires.*
RNG	*Générateur de nombres aléatoires.*
SIEM	*Security information and event management system.*
SSD	*Solid-state drive.*

Publications personnelles

[PBL+16] Aurélien Palisse, Hélène Le Bouder, Jean-Louis Lanet, Colas Le Guernic et Axel Legay, « Ransomware and the Legacy Crypto API », *in Risks and Security of Internet and Systems - 11th International Conference, CRiSIS 2016, Roscoff, France, September 5-7, 2016, Revised Selected Papers*, p. 11–28, 2016.

[PDB+17] Aurélien Palisse, Antoine Durand, Hélène Le Bouder, Colas Le Guernic et Jean-Louis Lanet, « Data Aware Defense (DaD) : Towards a Generic and Practical Ransomware Countermeasure », *in Secure IT Systems - 22nd Nordic Conference, NordSec 2017, Tartu, Estonia, November 8-10, 2017, Proceedings*, p. 192–208, 2017.

Bibliographie

[1] A Pseudorandom Number Sequence Test Program. Outil. URL : http://www.fourmilab.ch/random/.

[2] AutoIt. Outil de test d'interface graphique. URL : https://www.autoitscript.com/site/autoit/.

[3] BitBlaze. Plateforme d'analyse automatique de logiciels malveillants. URL : http://bitblaze.cs.berkeley.edu/.

[4] Bromium : Understanding Crypto-Ransomware. Rapport. URL : https://d3gv00hc5boujr.cloudfront.net/sites/default/files/rpt-bromium-crypto-ransomware-us-en.pdf.

[5] Cerber Ransomware 5.0 Released with a Few Changes. Blog. URL : https://www.bleepingcomputer.com/news/security/cerber-ransomware-5-0-released-with-a-few-changes/.

[6] CryptoAPI : fournisseur de services cryptographiques. Documentation. URL : https://docs.microsoft.com/en-us/windows/desktop/seccertenroll/cryptoapi-cryptographic-service-providers.

[7] CryptoAPI : Microsoft Enhanced Cryptographic Provider. Documentation. URL : https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp238.pdf.

[8] Cryptographic Provider Development Kit. Code. URL : https://www.microsoft.com/en-us/download/details.aspx?id=30688.

[9] CrystalDiskMark : filesystem benchmark. Outil. URL : http://crystalmark.info/software/CrystalDiskMark/index-e.html.

[10] Cuckoo Sandbox. Plateforme d'analyse automatique de logiciels malveillants. URL : https://cuckoosandbox.org/.

[11] Des ransomwares en pagaille au ministère des Transports. . URL : https://www.lemondeinformatique.fr/actualites/lire-des-ransomwares-en-pagaille-au-ministere-des-transports-63649.html.

[12] Does malware still detect virtual machines ? Blog. URL : https://www.symantec.com/connect/blogs/does-malware-still-detect-virtual-machines.

[13] Dynamic-Link Library Search Order. Documentation. URL : https://docs.microsoft.com/en-us/windows/desktop/dlls/dynamic-link-library-search-order.

[14] Firmware Analysis Tool. Outil. URL : https://github.com/ReFirmLabs/binwalk.

[15] Geekbench : New benchmarks, redesigned interface. Outil. URL : http://geekbench.com/.

[16] How NTFS Works. Documentation. URL : https://technet.microsoft.com/pt-pt/library/cc781134(v=ws.10).aspx.

[17] How to defend your website with ZIP bombs. Blog. URL : https://blog.haschek.at/2017/how-to-defend-your-website-with-zip-bombs.html.

[18] IDA. Désassembleur et décompilateur. URL : https://www.hex-rays.com/products/ida/.

[19] Introduction Intel AES-NI. Documentation. URL : https://software.intel.com/en-us/articles/introduction-to-intel-aes-ni-and-intel-secure-key-instructions.

[20] IOzone : filesystem benchmark. Outil. URL : http://www.iozone.org/.

[21] Kolmogorov-Smirnov Test. Documentation. URL : http://www.physics.csbsju.edu/stats/KS-test.html.

[22] Logiciel de rançon. Définition. URL : https://fr.wikipedia.org/wiki/Ran%C3%A7ongiciel.

[23] Logiciel malveillant. Définition. URL : https://fr.wikipedia.org/wiki/Logiciel_malveillant.

[24] Look Into Locky Ransomware. Blog. URL : https://blog.malwarebytes.com/threat-analysis/2016/03/look-into-locky/.

[25] Malekal. Base de données de logiciels malveillants. URL : http://malwaredb.malekal.com/.

[26] Malwr. Base de données de logiciels malveillants. URL : https://malwr.com/.

[27] Memory Limits for Windows and Windows Server Releases. Documentation. URL : https://docs.microsoft.com/en-us/windows/desktop/memory/memory-limits-for-windows-releases.

[28] Memory mapped files in a File System Filter Driver. Documentation. URL : https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/memory-mapped-files-in-a-file-system-filter-driver.

[29] Microsoft : File System Minifilter Drivers. Documentation. URL : https://msdn.microsoft.com/en-us/windows/hardware/drivers/ifs/file-system-minifilter-drivers.

[30] Minifilter Driver : Allocated Altitudes. Documentation. URL : https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/allocated-altitudes.

[31] Nvmtrace. Plateforme d'analyse automatique de logiciels malveillants. URL : https://github.com/adamwallred/nvmtrace.

[32] PCMark 8 The Complete Benchmark for Windows 8.1, Windows 8 and Windows 7. Outil. URL : https://www.futuremark.com/benchmarks/pcmark.

[33] Playing With Authenticode and MD5 Collisions. Blog. URL : https://blog.didierstevens.com/2009/01/17/playing-with-authenticode-and-md5-collisions/.

[34] Positive and Negative Affect Schedule (PANAS). Test. URL : https://booksite.elsevier.com/9780123745170/Chapter%203/Chapter_3_Worksheet_3.1.pdf.

[35] Real-time Automation to Discover, Detect and Alert of Ransomware (RADDAR). Plateforme d'analyse automatique de logiciels malveillants. URL : https://github.com/BUseclab/raddar.

[36] Scrapy. Extracteur de contenu de site Internet. URL : https://scrapy.org/.

[37] Stages of Adoption of Technology (SA). Test. URL : https://iittl.unt.edu/sites/default/files/Instruments/Stagesofadoption.pdf.

[38] TeslaCrypt 2.0 disguised as CryptoWall. Blog. URL : https://securelist.com/teslacrypt-2-0-disguised-as-cryptowall/71371/.

[39] The Talos Group : MBR Filter Driver. Outil. URL : https://github.com/vrtadmin/MBRFilter.

[40] Towards Generic Ransomware Detection. Blog. URL : https://objective-see.com/blog/blog_0x0F.html.

[41] VirtualBox. Logiciel libre de virtualisation. URL : https://www.virtualbox.org.

[42] VirusShare. Base de données de logiciels malveillants. URL : https://virusshare.com/.

[43] WinDbg : Debugging Tools for Windows. Outil. URL : https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools.

[44] Windows Performance Toolkit. Outil. URL : https://msdn.microsoft.com/en-us/windows/hardware/commercialize/test/wpt/index.

[45] Bulletin d'alerte Debian, DSA-1571-1 Openssl, générateur de nombres aléatoires prévisible. Rapport technique, 2008. URL : http://www.debian.org/security/2008/dsa-1571.

[46] Police Ransomware Threat Assessment. Rapport, 2014. URL : https://www.europol.europa.eu/sites/default/files/documents/policeransomware-threatassessment.pdf.

[47] Juniper Networks : CVE-2015-7755. Rapport technique, 2015. URL : https://kb.juniper.net/InfoCenter/index?page=content&id=JSA10713.

[48] Bulletin de sécurité AV-TEST. Rapport technique, 2016. URL : https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2015-2016.pdf.

[49] How to protect your networks from ransomware. , 2016. URL : https://www.justice.gov/criminal-ccips/file/872771/download.

[50] Bulletin de sécurité Kaspersky. Rapport technique, 2017. URL : https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/07164824/KSB_Story_of_the_Year_Ransomware_FINAL_eng.pdf.

[51] Chernobyl nuclear power plant hit by ransomware cyber attack. , 2017. URL : https://www.express.co.uk/news/world/821971/Chernobyl-nuclear-power-plant-hit-ransomware-cyber-attack.

[52] File System Minifilter Drivers. Documentation, Avril 2017. URL : https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/file-system-minifilter-drivers.

[53] NOTE D'INFORMATION DU CERT-FR : protection contre les rançongiciels. , 2017. URL : https://www.cert.ssi.gouv.fr/information/CERTFR-2017-INF-001/.

[54] Surprising Backup Failure Statistics that Justify Additional Protection. Rapport technique, 2017. URL : https://blog.barkly.com/backup-failure-statistics.

[55] WannaCry, Petya, NotPetya : how ransomware hit the big time in 2017. , 2017. URL : https://www.theguardian.com/technology/2017/dec/30/wannacry-petya-notpetya-ransomware.

[56] Cybercriminals Target Hospitals with SamSam Ransomware Attacks. , 2018. URL : https://healthitsecurity.com/news/cybercriminals-target-hospitals-with-samsam-ransomware-attacks.

[57] Internet users in the world. , 2018. URL : http://www.internetlivestats.com/internet-users/.

[58] Must-Know Ransomware Statistics 2018. , 2018. URL : https://blog.barkly.com/ransomware-statistics-2018.

[59] PE Format. Documentation, 2018. URL : https://docs.microsoft.com/fr-fr/windows/desktop/Debug/pe-format.

[60] SetWindowsHookExA function. Documentation, 2018. URL : https://docs.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-setwindowshookexa.

[61] Share of households with a computer at home worldwide from 2005 to 2017. , 2018. URL : https://www.statista.com/statistics/748551/worldwide-households-with-compute.

[62] The Week in Ransomware. Blog, juillet 2018. URL : https://www.bleepingcomputer.com/news/security/the-week-in-ransomware-july-27th-2018-ransomware-still-a-threat/.

[63] Nicoló Andronio, Stefano Zanero, and Federico Maggi. HelDroid : Dissecting and Detecting Mobile Ransomware. In Herbert Bos, Fabian Monrose, and Gregory Blanc, editors, *Research in Attacks, Intrusions, and Defenses*, pages 382–404, Cham, 2015. Springer International Publishing.

[64] Achiya Bar-On, Itai Dinur, Orr Dunkelman, Rani Hod, Nathan Keller, Eyal Ronen, and Adi Shamir. Optimal Backup Strategies Against Cyber Attacks. *CoRR*, abs/1704.02659, 2017. URL : http://arxiv.org/abs/1704.02659, arXiv:1704.02659.

[65] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM. URL : http://doi.acm.org/10.1145/945445.945462, doi:10.1145/945445.945462.

[66] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In USENIX Annual Technical Conference, FREENIX Track, 2005.

[67] Bill Blunden. *The Rootkit Arsenal : Escape and Evasion in the Dark Corners of the System.* Jones and Bartlett Publishers, Inc., USA, 2009.

[68] Alexei Bulazel and Bülent Yener. A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion : PC, Mobile, and Web. In *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*, ROOTS, pages 2 :1–2 :21, New York, NY, USA, 2017. ACM. URL : http://doi.acm.org/10.1145/3150376.3150378, doi:10.1145/3150376.3150378.

[69] Clonezilla. Logiciel libre et open source pour l'imagerie de disque et le clonage. URL : http://clonezilla.org/.

[70] Fred Cohen. Rogue Programs : Viruses, Worms and Trojan Horses. chapter Computer Viruses ; Theory and Experiments, pages 356–378. Van Nostrand Reinhold Co., New York, NY, USA, 1990. URL : http://dl.acm.org/citation.cfm?id=96745.98066.

[71] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. ShieldFS : A Self-healing, Ransomware-aware Filesystem. In *ACSAC*, pages 336–347. ACM, 2016.

[72] Digital Corpora. Bibliothèque de fichiers. URL : http://digitalcorpora.org/.

[73] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether : Malware Analysis via Hardware Virtualization Extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 51–62, New York, NY, USA, 2008. ACM. URL : http://doi.acm.org/10.1145/1455770.1455779, doi:10.1145/1455770.1455779.

[74] Patrick Dreyfus. Messages d'alerte pour la cybersécurité : effets des facteurs ergonomiques et contextuels sur la perception et le comportement des utilisateurs. Mémoire de Master 2, Laboratoire d'Observation des Usages des TIC, Université Rennes 2, 2018.

[75] Echraf Mekacher. Projet WARNING 1. Rapport technique, Laboratoire d'Observation des Usages des TIC, Université Rennes 2, 2018.

[76] Ronald A Fisher. Has Mendel's work been rediscovered ? *Annals of science*, 1(2) :115–137, 1936.

[77] Ronald Aylmer Fisher and Leonard Henry Caleb Tippett. Limiting forms of the frequency distribution of the largest or smallest member of a sample. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 24, pages 180–190. Cambridge University Press, 1928.

[78] Tal Garfinkel, Keith Adams, Andrew Warfield, and Jason Franklin. Compatibility is Not Transparency : VMM Detection Myths and Realities. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems*, HOTOS'07, pages 6 :1–6 :6, Berkeley, CA, USA, 2007. USENIX Association. URL : http://dl.acm.org/citation.cfm?id=1361397.1361403.

[79] Alexandre Gazet. Comparative analysis of various ransomware virii. *Journal in Computer Virology*, 6(1) :77–90, 2010. URL : https://doi.org/10.1007/s11416-008-0092-2, doi:10.1007/s11416-008-0092-2.

[80] Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. Next Generation Cryptographic Ransomware. In *Secure IT Systems - 23rd Nordic Conference, NordSec 2018, Oslo, Norway, November 28-30, 2018, Proceedings*, pages 385–401, 2018. URL : https://doi.org/10.1007/978-3-030-03638-6_24, doi:10.1007/978-3-030-03638-6_24.

[81] Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. No Random, No Ransom : A Key to Stop Cryptographic Ransomware. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 15th International Conference, DIMVA 2018, Saclay, France, June 28-29, 2018, Proceedings*, pages 234–255, 2018. URL : https://doi.org/10.1007/978-3-319-93411-2_11, doi:10.1007/978-3-319-93411-2_11.

[82] Julio Hernandez-Castro, Edward Cartwright, and Anna Stepanova. Economic Analysis of Ransomware. *CoRR*, abs/1703.06660, 2017. URL : http://arxiv.org/abs/1703.06660, arXiv:1703.06660.

[83] Danny Yuxing Huang, Maxwell Matthaios Aliapoulios, Vector Guo Li, Luca Invernizzi, Elie Bursztein, Kylie McRoberts, Jonathan Levin, Kirill Levchenko, Alex C. Snoeren, and Damon McCoy. Tracking Ransomware End-to-end. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 618–631, 2018. URL : https://doi.org/10.1109/SP.2018.00047, doi:10.1109/SP.2018.00047.

[84] Jian Huang, Jun Xu, Xinyu Xing, Peng Liu, and Moinuddin K. Qureshi. FlashGuard : Leveraging Intrinsic Flash Properties to Defend Against Encryption Ransomware. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2231–2244, New York, NY, USA, 2017. ACM. URL : http://doi.acm.org/10.1145/3133956.3134035, doi:10.1145/3133956.3134035.

[85] Galen Hunt and Doug Brubacher. Detours : Binary Interception of Win32 Functions. In *3rd USENIX Windows NT Symposium*, 1999.

[86] A. Kharraz, W. Robertson, and E. Kirda. Protecting against Ransomware : A New Line of Research or Restating Classic Ideas ? *IEEE Security Privacy*, 16(3) :103–107, May 2018. doi:10.1109/MSP.2018.2701165.

[87] Amin Kharraz, Sajjad Arshad, Collin Mulliner, William K. Robertson, and Engin Kirda. UNVEIL : A Large-Scale, Automated Approach to Detecting Ransomware. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 757–772, 2016. URL : https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kharaz.

[88] Amin Kharraz and Engin Kirda. Redemption : Real-Time Protection Against Ransomware at End-Hosts. In *Research in Attacks, Intrusions, and Defenses - 20th International Symposium, RAID 2017, Atlanta, GA, USA, September 18-20, 2017, Proceedings*, pages 98–119, 2017. URL : https://doi.org/10.1007/978-3-319-66332-6_5, doi:10.1007/978-3-319-66332-6_5.

[89] Amin Kharraz, William K. Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. Cutting the Gordian Knot : A Look Under the Hood of Ransomware Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 12th International Conference, DIMVA 2015, Milan, Italy, July 9-10, 2015, Proceedings*, pages 3–24, 2015. URL : https://doi.org/10.1007/978-3-319-20550-2_1, doi:10.1007/978-3-319-20550-2_1.

[90] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. Certified Malware : Measuring Breaches of Trust in the Windows Code-Signing PKI. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1435–1448, New York, NY, USA, 2017. ACM. URL : http://doi.acm.org/10.1145/3133956.3133958, doi:10.1145/3133956.3133958.

[91] James C. King. Symbolic Execution and Program Testing. *Commun. ACM*, 19(7) :385–394, July 1976. URL : http://doi.acm.org/10.1145/360248.360252, doi:10.1145/360248.360252.

[92] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. BareBox : Efficient Malware Analysis on Bare-metal. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 403–412, New York, NY, USA, 2011. ACM. URL : http://doi.acm.org/10.1145/2076732.2076790, doi:10.1145/2076732.2076790.

[93] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. PayBreak : Defense Against Cryptographic Ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab*

Emirates, April 2-6, 2017, pages 599–611, 2017. URL : http://doi.acm.org/10.1145/3052973.3053035, doi:10.1145/3052973.3053035.

[94] Kyungroul Lee, Insu Oh, and Kangbin Yim. Ransomware-Prevention Technique Using Key Backup. In Jason J. Jung and Pankoo Kim, editors, *Big Data Technologies and Applications*, pages 105–114. Springer International Publishing, 2017.

[95] Kyungroul Lee, Kangbin Yim, and Jung Taek Seo. Ransomware-Prevention Technique Using Key Backup. *Concurrency and Computation : Practice and Experience*, 30(3), 2017. URL : https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4337, doi:10.1002/cpe.4337.

[96] Faustin Mbol, Jean-Marc Robert, and Alireza Sadighian. An Efficient Approach to Detect TorrentLocker Ransomware in Computer Systems. In *CANS*, volume 10052 of *Lecture Notes in Computer Science*, pages 532–541, 2016.

[97] Shagufta Mehnaz, Anand Mudgerikar, and Elisa Bertino. RWGuard : A Real-Time Detection System Against Cryptographic Ransomware. In Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis, editors, *Research in Attacks, Intrusions, and Defenses*, pages 114–136, Cham, 2018. Springer International Publishing.

[98] Gregor Mendel. Versuche über Pflanzenhybriden. *Verhandlungen des naturforschenden Vereines in Brunn 4 : 3*, 44, 1866.

[99] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis. Spotless Sandboxes : Evading Malware Analysis Systems Using Wear-and-Tear Artifacts. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1009–1024, May 2017. doi:10.1109/SP.2017.42.

[100] Rajeev Nagar. *Windows NT File System Internals : A Developer's Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1997.

[101] Ana M Pires and Joao A Branco. A Statistical Model to Explain the Mendel—Fisher Controversy. *Statistical Science*, pages 545–565, 2010.

[102] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. v. Steen. Prudent Practices for Designing Malware Experiments : Status Quo and Outlook. In *2012 IEEE Symposium on Security and Privacy*, pages 65–79, May 2012. doi:10.1109/SP.2012.14.

[103] Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin R. B. Butler. CryptoLock (and Drop It) : Stopping Ransomware Attacks on User Data. In *ICDCS*, pages 303–312. IEEE Computer Society, 2016.

[104] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. AVclass : A Tool for Massive Malware Labeling. In Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquin Garcia-Alfaro, editors, *Research in Attacks, Intrusions, and Defenses*, pages 230–253, Cham, 2016. Springer International Publishing.

[105] Daniele Sgandurra, Luis Muñoz-González, Rabih Mohsen, and Emil C Lupu. Automated Dynamic Analysis of Ransomware : Benefits, Limitations and use for Detection. *arXiv preprint arXiv :1609.03020*, 2016.

[106] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouët. Anomaly Detection in Streams with Extreme Value Theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1067–1075, 2017. URL : http://doi.acm.org/10.1145/3097983.3098144, doi:10.1145/3097983.3098144.

[107] Michael Sikorski and Andrew Honig. *Practical Malware Analysis : The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition, 2012.

[108] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

[109] Teryl Taylor, Frederico Araujo, Anne Kohlbrenner, and Marc Ph. Stoecklin. Hidden in Plain Sight : Filesystem View Separation for Data Integrity and Deception. In Cristiano Giuffrida, Sébastien Bardin, and Gregory Blanc, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 256–278, Cham, 2018. Springer International Publishing.

[110] Viper. Outil de gestion et d'analyse des binaires. URL : http://viper.li/.

[111] Adam L. Young. Cryptoviral extortion using Microsoft's Crypto API. *International Journal of Information Security*, 5(2) :67–76, Apr 2006. URL : https://doi.org/10.1007/s10207-006-0082-7, doi:10.1007/s10207-006-0082-7.

[112] Adam L. Young and Moti Yung. Cryptovirology : Extortion-Based Security Threats and Countermeasures. In *1996 IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland, CA, USA*, pages 129–140, 1996. URL : https://doi.org/10.1109/SECPRI.1996.502676, doi:10.1109/SECPRI.1996.502676.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Analyse et détection de logiciels de rançon

Mot clés : logiciels malveillants, détection temps réel, outils statistiques

Resumé : La thèse s'intéresse aux logiciels de rançon, présente une plateforme d'analyse automatique et propose des contre-mesures. Nos contre-mesures sont conçues pour être temps réel et déployées sur une machine, c'est-à-dire "End-Hosts". En 2013 les logiciels de rançon font de nouveau parler d'eux, pour finalement devenir une des menaces les plus sérieuses à partir de 2015. Un état de l'art détaillé des contre-mesures existantes est fourni. On peut ainsi situer les contributions de cette thèse par rapport à la littérature. Nous présentons également une plateforme d'analyse automatique de logiciels malveillants composée de machines nues. L'objectif est de ne pas altérer le comportement des échantillons analysés. Une première contre-mesure basée sur l'utilisation d'une librairie cryptographique par les logiciels de rançon est proposée. Celle-ci peut être facilement contournée. Nous proposons donc une seconde contre-mesure générique et agnostique. Cette fois, des indicateurs de compromission sont utilisés pour analyser le comportement des processus sur le système de fichiers. Nous détaillons comment de manière empirique nous avons paramétré cette contre-mesure pour la rendre : utilisable et efficace. Un des challenges de cette thèse étant de faire concilier performance, taux de détection et un faible taux de faux positifs. Enfin, les résultats d'une expérience utilisateur sont présentés. Cette expérience analyse le comportement des utilisateurs face à une menace. En dernière partie, nous proposons des améliorations à nos contributions mais aussi des pistes à explorer.

Title : Analysis and detection of ransomware

Keywords : malware, real-time detection, statistics

Abstract : This phD thesis takes a look at ransomware, presents an autonomous malware analysis platform and proposes countermeasures against these types of attacks. Our countermeasures are real-time and are deployed on a machine (i.e., end-hosts). In 2013, the ransomware become

a hot subject of discussion again, before becoming one of the biggest cyberthreats beginning of 2015. A detailed state of the art for existing countermeasures is included in this thesis. This state of the art will help evaluate the contribution of this thesis in regards to the existing current publications. We will also present an autonomous malware analysis platform composed of bare-metal machines. Our aim is to avoid altering the behaviour of analysed samples. A first countermeasure based on the use of a cryptographic library is proposed, however it can easily be bypassed. It is why we propose a second generic and agnostic countermeasure. This time, compromission indicators are used to analyse the behaviour of process on the file system. We explain how we configured this countermeasure in an empiric way to make it useable and effective. One of the challenge of this thesis is to collate performance, detection rate and a small amount of false positive. To finish, results from a user experience are presented. This experience analyses the user's behaviour when faced with a threat. In the final part, I propose ways to enhance our contributions but also other avenues that could be explored.